SONDERHEFT 10 Lit. 12 000/hfl. 18,-/dkr. 68,- ÖS 100,-/sfr. 14,- DM 14,-

64'er Sonderheft für C 128-Fans

Markt&Technik

128er



Kurse für Einsteiger

- ★ Maschinensprache-Monitor: So arbeitet man mit TEDMON
- ★ Grafikbefehle durchleuchtet

Tips & Tricks zu CP/M

- ★ MBasic im Test
- ★ Druckeranpassungen für Wordstar, Multiplan, dBase
- ★ Wichtige Hilfen für Einsteiger und Fortgeschrittene

Hardware für Bastler

- ★ Kompletter C 128-Schaltplan
- ★ Eigene Programme im ROM
- ★ Komfortable IEEE-Schnittstelle zum Selbstbauen

Viele Listings zum Abtippen

Alle Programme auch auf Diskette erhältlich


Interface
Inklusive!
64ER ONLINE

C 128 – der Weg zum Profi

Der C 128 ist eine ausgezeichnete Mischung zwischen »Spiel-Computer« und Profimaschine. Durch den integrierten C 64 stehen Ihnen die unendlich vielen Programme des meistverkauften Computers der Welt zur Verfügung. Der C 128-Modus dagegen ist eine gelungene Weiterentwicklung des C 64: Wesentlich besseres und schnelleres Basic, doppelt so viel Speicherplatz und im Profibereich übliche 80 Zeichen pro Bildschirmzeile. Damit werden Programme möglich, die den Vergleich zu den viel teureren sogenannten Personal Computern, sprich IBM und ähnliche, nicht zu scheuen brauchen und sogar übertreffen. Beispiele sind Protext, Vizawrite Classic oder Vizastar 128: viel schneller als zum Beispiel Word oder Lotus 1-2-3 und vor allem wesentlich billiger. Sie haben also schon eine gute Wahl getroffen.

Interessant ist auch der dritte Computer im C 128, der CP/M-Modus. Ein Vorteil ist, daß es viele bewährte Programme gibt, die auch auf anderen CP/M-Computern laufen. Ein weiterer, daß der Einstieg in die PC-, sprich IBM-Welt, erleichtert wird. Standard-Software, wie zum Beispiel Wordstar, dBase oder Multiplan, gibt es auch für die »Großen«, wenn auch um ein Vielfaches teurer. In immer mehr Firmen und Büros werden diese Programme genutzt. Sie haben also mit dem C 128 und seinem CP/M-Modus den großen Vorteil, schon mal zu üben und fit zu werden und zwar zu einem vernünftigen Preis. Auch in diesem Sonderheft finden Sie eine große Menge an Tips&Tricks zu Wordstar, dBase, Multiplan und CP/M überhaupt. Wenn Sie nicht oder nicht genau wissen, was CP/M eigentlich ist, klären wir Sie darüber ausführlich auf. CP/M ist aber nicht nur für Anwender-Software gut, sondern ebenfalls für Programmiersprachen wie Pascal, Microsoft-Basic oder CBasic. Diese Basic-Varianten gibt es jetzt auch für den C 128. Wir stellen beide in einem Test vor.

Ein echter Knüller wartet auf die Hardware-Freunde unter Ihnen: Der komplette C 128-Schaltplan, erstmalig in diesem 64'er-Sonderheft abgedruckt. Für Bastler ist er sicherlich unentbehrlich und wenn der Computer mal streiken sollte, ist die Fehlerquelle schneller eingegrenzt. Einen Bastelvorschlag wollen wir Ihnen nicht vorenthalten: Wer seinen C 128 schon einmal geöffnet hat, wird der freie Steckplatz aufgefallen sein. Wir sagen Ihnen, was Sie damit machen können und vor allem, wie.

Ein Megabyte Speicherkapazität pro Diskette ist schon fantastisch. Wer hat nicht schon mal mit dem Diskettenlaufwerk SFD 1000 geliebäugelt? Vielleicht haben Sie aber auch Gelegenheit, an ein Doppellaufwerk der alten Commodore-8000er-Serie heranzukommen. In jedem Fall haben Sie jedoch Anschlußprobleme mit dem C 128. Ihm fehlt eine IEEE-Schnittstelle. In diesem Sonderheft präsentieren wir Ihnen ein praxiserprobtes IEEE-Interface zum Selbstbauen. Mit diesem Interface, einem C 64 und einer SFD 1000 wird schon seit längerer Zeit eine umfangreiche Mailbox betrieben und zwar ohne Probleme.

Maschinensprache oder Assembler ist für einige das Nonplusultra, für andere eine Herausforderung und sehr viele haben noch keine Erfahrung damit. Der C 128 besitzt jedoch bekanntermaßen einen Maschinensprache-Monitor, der die ersten Schritte in Richtung Maschinensprache gewaltig erleichtert. Der TEDMON ist jedoch mehr als eine Hilfe beim Programmieren. Mit einem Monitor sollte auch ein Nur-Basic-Programmierer umgehen können. Warum und vor allem wie erklären wir Ihnen in einem sehr ausführlichen Kurs mit vielen Beispielen zum Experimentieren. Im Commodore-Handbuch wird zwar die Bedienung des TEDMON beschrieben, nicht jedoch die großartigen Möglichkeiten, die man mit ihm realisieren kann.

Wenn Sie Interesse an Listings zum Abtippen haben, werden Sie voll auf Ihre Kosten kommen. Alle in dieser Ausgabe veröffentlichten Basic- und Maschinensprache-Programme sind für den C 128-Modus gedacht, darunter ein sehr schnelles Kopierprogramm für die Floppy 1571.

Etwas Besonderes gibt es für Pascal-Freunde: Das abgedruckte Turbo-Pascal-Programm stellt die wichtigsten Informationen eines beliebigen Pascal-Programms in der Art einer Crossreferenzliste zusammen.

Wir haben auch mit diesem Sonderheft wieder versucht, Ihnen möglichst viele Informationen, Tips&Tricks sowie Listings zum Abtippen zu geben. Sicherlich können wir nicht alle Probleme klären und einige Fragen sind offen geblieben. Schreiben Sie uns, was Ihnen gefallen hat, was fehlt oder was wir das nächste Mal besser machen können!

Georg Klinge, leitender Redakteur

PROGRAMM-SERVICE



Besuchen Sie Markt&Technik auf folgenden Messen:

Buchmesse, Frankfurt 1.–6.10.86, Halle 6.0, Stand B 430
Interkama, Düsseldorf 6.–11.10.86, Halle 12, Stand C 27
Orgatechnik, Köln 16.–21.10.86, Halle 3.2 Stand K 29
Systec, München 27.–30.10.86, Halle 7 Stand D 1
Electronica, München 11.–15.11.86, Halle 18 Stand A 10

Markt&Technik Verlag AG, Hans-Pinsel-Str. 2, 8013 Haar bei München

Bestellungen in der Schweiz: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Tel. 042/41 56 56
Bestellungen in Österreich: Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Tel. 0222/83 31 96,
Microcomput-ique E. Schiller, Fasangasse 21, A-1030 Wien, Tel. 0222/78 56 61,
Ueberreuter Media Handels- und Verlagsgesellschaft mbH, Alser Straße 24, A-1091 Wien, Tel. 0222/48 15 38-0
Bestellungen aus anderen Ländern bitte per Auslandspostanweisung!

Das Angebot dieser Ausgabe:

Butler – Betriebssystemerweiterung für den C128

Butler ermöglicht ein komfortableres Arbeiten mit dem C128: Die Tastaturfunktionen werden erweitert und das ohnehin umfangreiche Basic 7.0 erhält neue, nützliche Befehle. So existieren weitere ESC-Funktionen genauso wie Anweisungen, die den Speicher direkt beeinflussen. Auch OLD fehlt nicht. Der Clou aber ist ein Befehl, mit dem sich der Z80 im C128-Modus ansprechen läßt.

1 Diskette für Commodore 128
Bestell-Nr. L6 86 S10 D (sFr. 24,90/öS 299,-*) DM 29,90*
* inkl. MwSt. Unverbindliche Preisempfehlung

Programme aus früheren Ausgaben:

64'er-Ausgabe 9/86
Bestell-Nr. L6 86 09D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
FSD64 (Listing des Monats)
Im 64'er-Modus mit der 1570/71 schnell laden (brennfähig) S. 50
Ean-Barcodes (Anwendung des Monats) Bar-Codes mit dem Epson-Drucker S. 54
Shrinksprite, Sprites drehen S. 58
Tips&Tricks zum C128 S. 62
Tips&Tricks für Einsteiger S. 64
Tips&Tricks für Profis S. 66
Cross-Reference für den C128 S. 71
Variablen-Dump für Anspruchsvolle S. 75
Neue Module für Hypra-Basic S. 76
Hi-Eddi mit dem Star NL-10 und dem GP 700 VC S. 79
HiRes Colossal (4 Grafikbildschirme unter Simons Basic) S. 82
Hardcopy für den 1520 S. 93
Der erste Druckerspeeder für den MPS 801 S. 94
Hardmaker für Epson-Drucker S. 95
Pascal-Kurs für Einsteiger (3 Listings) S. 137
Tips&Tricks zu Vizawrite (Teil 9) S. 159

64'er-Ausgabe 8/86
Bestell-Nr. L6 86 08D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
64'er Comal Sonderservice: COMAL 0.14 S. 42
Vokabel Trainer – ein geduldiger und preiswerter Englisch-Nachhilfe-Lehrer (AdM) S. 53
Digi-Controller (LdM) Simulieren und Austesten digitaler Schaltungen und speicherprogrammierbarer Steuerungen S. 59
Grafik Wandler S. 66
Reise durch den C128 S. 70
Tips&Tricks zum C16/C116/Plus 4 S. 74
Tips&Tricks für Einsteiger S. 75
Tips&Tricks für Profis S. 80
MPS Support für den MPS 802 S. 91
Hypra-Basic wird strukturiert S. 92
Comalchen für den C128 S. 128
Streifzüge durch die Grafikwelt S. 136
Tips&Tricks zu Superbase S. 149

64'er-Ausgabe 7/86
Bestell-Nr. L6 86 07D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Die Wachstumspyramide S. 22
Unvergleichbare Rhythmusmaschine (AdM) S. 52
Variosystem druckt für Sie (LdM) S. 56
Druckertreiber und Zeichensatzeditor zu Master-Text für Epson und MPS802 S. 67
Vectors: ein fesselndes Spiel für C128 S. 73
Reset-Schutz für Basic-Programme S. 78
Basic-Erweiterungen durchschaut S. 80
Hilfe für Schachspieler S. 81
Newsroom druckt Deutsch S. 89
Neues von Hypra-Basic S. 96

64'er-Ausgabe 6/86
Bestell-Nr. L6 86 06D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 5/86
Bestell-Nr. L6 86 05D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 4/86
Bestell-Nr. L6 86 04D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 3/86
Bestell-Nr. L6 86 03D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 2/86
Bestell-Nr. L6 86 02D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 1/86
Bestell-Nr. L6 86 01D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 12/85
Bestell-Nr. L6 85 12D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 85 12K Kassette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 11/85
Bestell-Nr. L6 85 11A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 10/85
Bestell-Nr. L6 85 10A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 9/85
Bestell-Nr. L6 85 09A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 8/85
Bestell-Nr. L6 85 08A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 7/85
Bestell-Nr. L6 85 07A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 6/85
Bestell-Nr. L6 85 06A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 5/85
Bestell-Nr. L6 85 05A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 4/85
Bestell-Nr. L6 85 04A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 3/85
Bestell-Nr. L6 85 03A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 2/85
Bestell-Nr. L6 85 02A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 1/85
Bestell-Nr. L6 85 01A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Sonderhefte

Sonderheft 9/86 – Floppy&Dateiverwaltung
Bestell-Nr. L6 86 S9 CD
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 8/86 – Plus/4 und C16
Bestell-Nr. L6 86 S8 CD Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 86 S8 KC 4 Kassetten
DM 34,90* (sFr. 29,50/öS 349,-*)
Bestell-Nr. L6 86 S8 KV Kassette
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 7/86 – PEEKs & POKEs
Bestell-Nr. L6 86 S7D 1 Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 6/86 – Grafik
2 Disketten mit allen Programmen
Bestell-Nr. L6 86 S6D1
DM 34,90* (sFr. 29,50/öS 349,-*)
1 Diskette mit Giga-CAD-Demos
Bestell-Nr. L6 86 S6D2
DM 19,90* (sFr. 17,-/öS 199,-*)
3 Disketten mit allen Programmen und Demos
Bestell-Nr. L6 86 S6D3
DM 49,80* (sFr. 43,50/öS 498,-*)

Sonderheft 5/86 – Grundwissen
Bestell-Nr. L6 86 S5D 1 Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 4/86 – Abenteuer
Bestell-Nr. L6 86 S4D 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 3/86 – C16, C116, VC20, Plus/4
1 Diskette für VC20 und C16/116:
Bestell-Nr. L6 86 S3 CD
DM 29,90* (sFr. 24,90/öS 299,-*)
1 Kassette für VC20:
Bestell-Nr. L6 86 S3 KV
DM 19,90* (sFr. 17,-/öS 199,-*)
1 Kassette für C16:
Bestell-Nr. L6 86 S3 KC
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 2/86 – Tips&Tricks
Bestell-Nr. L6 86 S2D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 1/86 – C128er
Bestell-Nr. L6 86 S1D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 8/85 – Assembler
Bestell-Nr. L6 85 S8D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 85 S8K Kassette
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 7/85 – Professionelle Anwendungen
Bestell-Nr. L6 85 S7D 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)
Bestell-Nr. L6 85 S7K 4 Kassetten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 6/85 – Top-Themen
Bestell-Nr. L6 85 S6 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 5/85 – Floppy, Datasette
Bestell-Nr. L6 85 S5D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 85 S5K Kassette
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 4/85 – Grafik
Bestell-Nr. L6 85 S4A
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 3/85 – Spiele
Bestell-Nr. L6 85 S3 A 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 2/85 – Abenteuerspiele
Bestell-Nr. L6 85 S2
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 1/85 – Tips&Tricks (2. überarb. Auflage)
Bestell-Nr. CB 023 Floppy-Utilities
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. CB 024 Hilfsprogramme
DM 29,90* (sFr. 24,90/öS 299,-*)

* inkl. MwSt. Unverbindliche Preisempfehlung.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die eingeheftete Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungs-Scheck mit Ihrer Bestellung. Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

INHALT

Fragen und Antworten zum C 128

Probleme mit C 128

Als Besitzer eines neuen C128 bin ich auf zwei Probleme gestoßen, bei denen ich vermute, daß mein C128 nicht ganz in Ordnung sein könnte:

(1) Die programmierte Umschaltung auf die DIN-Tastatur mittels »POKE 1,0« (siehe Handbuch, Kapitel 4.1, Seite 2) klappt bei mir nicht. Gibt es eine andere Möglichkeit zur Tastaturumschaltung?

(2) Zeichne ich im hochauflösenden Grafikmodus zum Beispiel zwei Kreise in einer bestimmten Farbe, und verbinde diese dann mit einer Linie in einer anderen Farbe, so werden diese verschiedenen Farben nicht voneinander getrennt, sondern es wird gleich das ganze Umfeld der Schnittstelle neu eingefärbt. Was kann ich dagegen tun?

Oliver Hobert

Zunächst eine gute Nachricht: Ihr C 128 ist, zumindest was die beschriebenen Symptome angeht, völlig in Ordnung.

(1) Die programmgesteuerte Tastaturumschaltung funktioniert auch bei Ihrem C 128. Es handelt sich hier um einen Fehler im Handbuch. Versuchen Sie die Umschaltung mit »POKE 0, PEEK(0) AND 64:POKE 1,0«.

(2) Dieser Effekt hat eine verblüffend einfache Ursache: Im hochauflösenden Grafikmodus kann der C 128 zwar einzelne Grafikpunkte setzen oder löschen, es ist jedoch nicht möglich, jedem Punkt eine individuelle Farbe zu geben. Die Farbgebung im hochauflösenden Modus entspricht genau derjenigen im Textmodus, das heißt, es kann jede Zeichenposition (25 x 40 Positionen auf dem Bildschirm) eine Farbe annehmen. Mit anderen Worten: Die gewählte Farbe kann nicht punktweise, sondern nur pro 8 x 8-Punkte-Bereich (eben gerade eine Zeichenposition) frei gewählt werden. Innerhalb einer 8 x 8-Zeichenmatrix haben immer alle 64 Grafikpunkte die gleiche Farbe. Ihr spezielles Problem läßt sich nur durch Verwendung des Multicolormodus lösen. In diesem Vielfarbenmodus kann jeder einzelne Grafikpunkt eine von vier möglichen Farben haben. Allerdings ist die Auflösung in diesem Modus mit 160 x 200 Punkten nur noch halb so groß wie im hochauflösenden Grafikmodus.

Was kann die 1571-Floppy?

(1) Können eine 1541 und eine 1571 gleichzeitig am C128 betrieben werden, um C64-Dateien auf das 1571-Format zu überspielen?

(2) Ist das C128-CP/M kompatibel zum C64-CP/M, das heißt, können CP/M-Programme vom C64 auch auf dem C128 laufen?

(3) Wie können Dateien des C64-CP/M auf das Format der 1571 gebracht werden?

(4) Kann man beim C128 unter CP/M auch die alte 1541 benutzen?

Dietmar Hoffman

(1) Natürlich, es ist kein Problem, beide Laufwerke gleichzeitig zu betreiben. Sie brauchen dazu nur die Gerätenummer einer der beiden Laufwerke von 8 auf 9 zu ändern.

(2) Das CP/M Plus des C 128 ist kompatibel zum CP/M 2.2, das auf dem C 64 mit CP/M-Modul läuft. Bei einigen Programmen (wie Textverarbeitung etc.) kann eine neue Anpassung mittels eines speziellen (immer mitgelieferten) Install-Programms notwendig werden.

(3) Zum Überspielen vom 1541-Format auf das 1571-Laufwerk formatiert man eine Diskette im 1571-Format, dann die alte Diskette darauf kopieren – fertig.

(4) Das CP/M des C 128 läuft auch mit der 1541. Eine CP/M-Diskette im 1541-Format liegt jedem C 128 bei.

Fragen zum C 128

(1) Gibt es für den C128 ein Grafikprogramm, bei dem man die gezeichneten Bilder über »BLOAD« laden und in eigenen Programmen verwenden kann?

(2) Kann man beim C128 mehrere Windows definieren, oder wird bei Definition eines weiteren Windows das alte gelöscht?

(3) Wie hebt man eine Window-Definition wieder auf?

(4) Kann man in Basic (ohne POKEs) Sprites um bestimmte Winkel drehen?

(5) Kann man Sprites um beliebig viele Punkte vergrößern oder verkleinern?

(6) Wenn ein Sprite mit der PAINT-Anweisung ausgefüllt und anschließend bewegt wird, bleibt das Sprite dann ausgefüllt oder bleibt die ausgefüllte Fläche am gleichen Ort zurück?

(7) Im Markt&Technik-Buch »Das C128-Handbuch« ist ein Listing für Grafik auf dem 80-Zeichen-Bildschirm abgedruckt. Ich weiß aber nicht die Adressen dieses Programms und kann es daher auch nicht mit »BSAVE« speichern.

(8) Wie kommt man (in Basic) an den zweiten Schreib-/Lesekopf der 1571-Floppy?

Thomas Weberstaedt

(1) Ein spezielles Grafikprogramm für den C 128-Modus ist uns bislang noch nicht bekannt. Sie können aber einfach ein entsprechendes Programm für den C 64 (im C 64-Modus) benutzen, um Ihre Bilder zu malen. Diese Bilder lassen sich dann völlig problemlos auch im C 128-Modus laden (die Grafik ist in beiden Betriebsarten die gleiche). Nähere Hinweise dazu erhalten Sie entweder im Handbuch zum Grafikprogramm oder (hoffentlich) auf Anfrage beim Hersteller.

(2) Nein, sobald ein Fenster (Window) definiert wird, ist die vorherige Einstellung vergessen.

(3) Im Direktmodus wird die Window-Definition durch zweimaligen Druck auf die <HOME>-Taste, im Programm am einfachsten durch Definition eines neuen Windows (das natürlich auch den gesamten Bildschirm umfassen kann) gelöscht.

(4) Das ist nicht möglich.

(5) Nur die bekannte Verdopplung der Sprite-Größe in X- und Y-Richtung ist möglich.

(6) Die PAINT-Anweisung bezieht sich nur auf die hochauflösende Grafik. Es ist damit nicht möglich, Sprites auszufüllen.

(7) Bei dem abgedruckten Listing handelt es sich um ein Assembler-Programm. Die vierstellige Zahl am Anfang jeder Zeile ist die (hexadezimale) Adresse jedes Befehls. Das Programm wird mit dem eingebauten Maschinensprache-Monitor des C 128 eingegeben und gespeichert. Die Anfangs- und Endadresse des Programms ist einfach die Adresse des ersten und des letzten Befehls, geht also von $1400 bis $157F.

(8) Der Zugriff auf die eine oder andere Diskettenseite wird vom DOS der 1571 selbständig geregelt. Sie brauchen keine besonderen Vorkehrungen zu treffen, um einen Schreib-/Lesekopf auszuwählen.

Kopieren mit 1541 und 1571?

Kann man mit einer 1541-Floppy und dem neuen 1571-Laufwerk Disketten von einem Laufwerk zum anderen kopieren? Funktionieren alle Kopierprogramme auch mit der 1571?

Dieter Stahlhofer

Die 1571 kann Disketten der 1541 sowohl lesen als auch selbst in diesem Format beschreiben. Das Kopieren mit zwei Laufwerken ist prinzipiell möglich. Sie müssen aber einen wichtigen Punkt beachten: Wenn Sie den C 128 in den C 64-Modus schalten, geht die 1571-Floppy automatisch in den 1541-Modus. Leider ist die 1571 nur eingeschränkt kompatibel zur 1541. Die meisten Kopierprogramme (insbesondere die schnellsten) laufen nicht mit der 1571. Dasselbe gilt übrigens auch für kopiergeschützte C 64-Software. Die meisten kopiergeschützten Programme laufen auf einer 1571 nicht. Das führt dann zu der oft merkwürdigen Situation, daß Besitzer von (teuren) Originalen beim Umstieg vom C 64 zum C 128 Ihre Software nicht mehr verwenden können.

RS232C und CP/M?

Beim C128 gibt es unter CP/M Probleme, die RS232C-Schnittstelle anzusprechen, die im C128- und im C64-Modus einwandrei funktioniert. Unter CP/M wird bei Aufruf der entsprechenden Routinen zwar die Ausgabe verlangsamt, aber an keinem einzigen Pin des User Ports erscheint ein Signal.

Manfred Kramer

Da es unter CP/M anscheinend Probleme mit der seriellen Schnittstelle gab, hat Commodore die entsprechenden CP/M-Routinen einfach mit einem RET-Opcode »gesperrt«. Die seriellen CP/M-Routinen lassen sich zwar noch aufrufen, aber kehren ganz einfach unverrichteter Dinge wieder zurück.

C128 ohne Commodore-Monitor?

Welche Lösung gibt es, wenn man den C128 auf 80 Zeichen bringen will, ohne den Original-Monitor von Commodore verwenden zu wollen? Welche Firma stellt eine entsprechende Kabelverbindung her?

Bernd Becks

Leider ist Ihr Problem auch nicht so generell zu lösen, denn es gibt eine ganze Zahl verschiedener Steckernormen. Außerdem wollen Sie ja bestimmt nicht auf den 40-Zeichen-Modus verzichten, so daß auf jeden Fall eine Spezialschaltung notwendig wird, da der C128 bekanntlich zwei völlig unterschiedliche Videonormen für 40 und 80 Zeichen verwendet. Falls Ihnen ein monochromer Monitor ausreicht, können Sie die in unserem »128'er«-Sonderheft, Ausgabe 1/86, abgedruckte Schaltung verwenden, mit der Sie von der Computertastatur aus zwischen 40 und 80 Zeichen umschalten können. Eine Übersicht über Farbmonitore auch für den C128 finden Sie in der Ausgabe 1/86 des 64'er-Magazins. Welche Kabelverbindung Sie benötigen, richtet sich nicht zuletzt auch nach dem von Ihnen gewählten Monitormodell. Sie müssen sich also zuerst für einen Monitor entscheiden, ehe Sie sich ein Kabel kaufen können.

MPS 801 am C128?

Kann der Commodore-Drucker MPS 801 am C128 in allen drei Betriebsarten (C64, C128, CP/M) ohne Interface betrieben werden? Die befragten Commodore-Händler beantworteten diese Frage zu jeweils gleichen Teilen mit »ja« und »nein«.

Iris und Joachim Mallach

Alle Commodore-Drucker arbeiten in allen drei Betriebsarten einwandfrei mit dem C128 zusammen. Da Commodore sich jedoch beim Zeichensatz seiner Computer (und Drucker) an keinerlei Normen wie ASCII oder DIN gebunden hat, treten Probleme beim DIN-Modus des C128 auf. Im DIN-Modus ist der Zeichensatz nämlich entsprechend der DIN-Norm codiert (also Standard-ASCII-Zeichen plus deutsche Sonderzeichen). Alle MPS-Drucker verstehen aber nur die »Commodore-Norm«, so daß Sie den C128 beim Betrieb mit einem MPS-Drucker im DIN-Modus nicht optimal verwenden können. Anders sieht die Sache bei CP/M aus. Auf der Systemdiskette befindet sich ein Programm namens »SETUP.COM«, mit dem Sie das CP/M-System an verschiedene Druckertypen anpassen können. Wenn Sie einen Commodore-Drucker verwenden, wählen Sie bei diesem Programm einfach die entsprechende Option (»C« für Commodore) aus und können dann sogar, falls Ihr Drucker dazu fähig ist, deutsche Umlaute zu Papier bringen. Ein Interface für Commodore-Drucker ist nicht erforderlich.

Schwierigkeiten mit dem C128

(1) Manche Spiele laufen im C64-Modus nicht richtig. Der Bildschirm ist dann mit vielen grafikähnlichen Zeichen beschrieben und es tut sich rein gar nichts mehr.

(2) Der MSE 1.0 aus der 64'er funktioniert bei mir nicht. Er ignoriert alle Tasten außer Q, Z, 2 und der Leertaste.

(3) Nach einem Reset bleiben Maschinensprache-Programme offenbar im Speicher stehen und werden nicht gelöscht. Ist dieser Umstand normal?

(4) Warum kann der C128 Programme für den C64 trotz seiner sprachlichen Kompatibilität nicht auch im C128-Modus direkt ausführen?

(5) Ist es normal, daß im 40-Zeichen-Modus bei bestimmten Farbkombinationen die Buchstaben verwischt und unleserlich werden?

(6) In verschiedenen Artikeln ist von einer RS232C-Schnittstelle über den User Port die Rede. Ich brauche eine solche Schnittstelle zum Anschluß meines Druckers, habe aber kein Programm, um den User Port ansprechen zu können.

Fritz Liedebach

(1) Es ist möglich, daß einige Spiele infolge des verwendeten Kopierschutzes nicht richtig mit einem 1570/1571-Laufwerk geladen werden können.

(2) Entweder haben Sie den MSE falsch abgetippt, oder Sie betreiben das Programm irrtümlich im C128-Modus (was nicht funktionieren kann). Der MSE ist auf jeden Fall ohne Fehler.

(3) Bei einem Reset wird nur der Computer neu initialisiert, Machinenprogramme bleiben aber in der Regel erhalten.

(4) Der Grund hierfür ist die völlig andere Speicherverwaltung (Bankswitching) im C128-Modus gegenüber dem C64. Die meisten POKE-, PEEK- und SYS-Befehle des C64 funktionieren daher nicht im C128-Modus.

(5) Bei einigen Farbkombinationen ist der Monitor einfach überfordert: Die Zeichen werden verschwommen und undeutlich dargestellt. Dies ist völlig normal und kein Grund zur Beunruhigung.

(6) Um die RS232C-Schnittstelle anzusprechen, brauchen Sie keine zusätzliche Software. Diese ist bereits im Betriebssystem vorhanden. In der Ausgabe 5/86 des 64'er-Magazins finden Sie nähere Informationen über die RS 232C-Schnittstelle des C64, die mit der des C128 identisch ist.

CP/M-Probleme beim C128

Ich arbeite seit einiger Zeit mit Wordstar 3.0 für den C128. Beim Ausdrucken erscheinen nun stets die beiden letzten Zeichen einer Textzeile in der nächsten Druckzeile und zwar am äußersten linken Rand, während eine normale Textzeile etwas weiter rechts beginnt. Ich habe festgestellt, daß man diesen Effekt verhindern kann, indem man die rechte Randeinstellung auf 70 Zeichen begrenzt. Allerdings scheint mir das doch recht unkomfortabel zu sein.

Volker Zietek

Wordstar fängt beim Ausdruck nicht am äußersten linken Rand an, sondern läßt links und rechts vom Text etwas Raum frei. Dadurch hat man beim Ausdruck nicht die vollen 80 Zeichen zur Verfügung, sondern weniger. Durch diese Eigenschaft von Wordstar wird das Erscheinungsbild einer gedruckten Seite stark verbessert und ähnelt mehr einer konventionellen Schreibmaschine, die ja auch nur etwa 60 bis 65 Zeichen in eine Zeile bringt. Wenn man unbedingt die volle Druckbreite ausnutzen möchte, kann man den linken und rechten Rand auch abschalten. Hierzu verwendet man das Installationsprogramm. Nähere Angaben hierzu gibt das Handbuch.

MS-DOS für den C128

Seit kurzem besitze ich den C128 mit CP/M-Betriebssystem. Da ich ihn hauptsächlich gewerblich nutze, ist natürlich das Betriebssystem MS-DOS für mich interessanter. Ich möchte daher fragen, ob es irgendwann eine MS-DOS-Systemdiskette für den C128 geben wird?

Michael Kundler

Das wird es ganz sicher nicht geben, denn CP/M ist ein Betriebssystem für den Z80-Prozessor von Zilog, während MS-DOS ein Betriebssystem für den 8086 von Intel ist. Um MS-DOS betreiben zu können, brauchen Sie einen IBM-kompatiblen PC; auf dem C128 ist es jedenfalls nicht möglich.

Apple-CP/M auf dem C128?

Kann man die CP/M-Version 2.20B für den Apple II plus auch auf dem C128 laufen lassen?

Santosh C. Purakal



Wenn Sie damit etwa meinen sollten, ob man eine CP/M-Diskette des Apple auf dem C 128 booten kann, dann ist die Antwort auf jeden Fall »nein«. Sie können generell niemals die CP/M-Systemdisketten eines Computers auf einem anderen System laufen lassen, denn das CP/M-System dient gerade dazu, die völlig unterschiedliche Hardware der einzelnen Computer per Software kompatibel zu machen. Das bedeutet aber, daß das CP/M-System selbst natürlich geräteabhängig ist. Für jedes Computermodell muß das CP/M-System daher speziell angepaßt werden.

Wenn Sie allerdings meinen, ob CP/M-Programme von Apple auch auf dem C 128 lauffähig sind, kann man dazu nur folgendes sagen: Alle CP/M-Programme, egal für welchen Computer, laufen auf dem C 128 – es sei denn, es handelt sich um solche, die unter Umgehung des CP/M-Standards direkt bestimmte Hardwareeigenschaften eines Computers ansprechen. Um ein beliebiges CP/M-Programm eines anderen Computers auf dem C 128 laufen zu lassen, müssen Sie es nur auf eine C 128-Systemdiskette kopieren.

Bei Programmen, die eine anspruchsvollere Bildschirmausgabe benötigen (Textverarbeitungsprogramme, etc.) ist allerdings in den meisten Fällen noch eine spezielle Installation notwendig. Das genaue Vorgehen in diesen Fällen ist in den jeweiligen Handbüchern zu diesen Programmen beschrieben.

Speicherprobleme beim C 128?

Ich habe ein Statik-Programm für den C128 entwickelt, das folgende Problematik aufweist: Das Programm selbst hat eine Länge von 16,5 KByte, so daß in der Speicherbank 0 noch über 40 KByte eigentlich ungenutzt verbleiben. Das Programm benötigt insgesamt aber etwa 100 KByte für Variable (einfache Variable und Felder). Wie mir bekannt ist, werden diese Variable in der Speicherbank 1 verwaltet. Allerdings sind hier effektiv nur 62 KByte für Variable frei. Mir fehlen also rund 40 KByte an Variablenspeicher, die aber in Bank 0 verfügbar wären. Wie lassen sich also Speicher-Bank 0 und 1 von Basic aus manipulieren, so daß dem Basic-Programm eben nur soviel Speicherplatz zukommt, wie es effektiv benötigt, der restliche Speicherbereich (sowohl von Bank 0 als auch von Bank 1) aber voll für Variable zur Verfügung steht?

Wolfgang Schwenkglenks

Dieses Problem wäre nur durch eine umfangreiche Basic-Erweiterung zu lösen, die wesentlich häufiger von dem zeitraubenden Verfahren des Bankswitching Gebrauch macht, als es das Basic 7.0 tut. Eine einfache Lösung für Ihr Problem (etwa ein paar POKE-Befehle oder ähnliches) ist auf jeden Fall nicht möglich.

Sie sollten sich vielleicht eher einmal überlegen, ob es nicht möglich ist, einige besonders umfangreiche Feldvariable in Form einer sequentiellen oder relativen Datei auf Diskette auszulagern und somit Speicherplatz zu sparen.

Funktionstasten

Im C 64-Modus meines C128 kann ich die Funktionstasten mit Hilfe des Befehls »GET« abfragen. Da im C128-Modus die Funktionstasten mit Befehlen belegt sind, erhalte ich immer nur die »CHR$«-Codes für die Basic-Wörter. Wie kann ich die Funktionstasten auch im C128-Mode effektiv abfragen?

Hans Schnurr

Im C 128-Modus können Sie mit den Funktionstasten viel professioneller arbeiten als im C 64-Modus. Der nötige Befehl lautet »KEY«. Damit können Sie die Belegung der Funktionstasten am Bildschirm anzeigen oder ändern (lesen Sie dazu das entsprechende Kapitel in Ihrem Handbuch nach). Wenn Sie bei der Definition der Funktionstasten an das Ende des entsprechenden Ausdrucks ein »CHR$(13)« anhängen, läßt sich die Funktionstaste auch mit dem »INPUT«-Befehl verwenden. Man drückt die entsprechende Funktionstaste und schon ist die Variable mit dem String der Funktionstaste belegt. Beim Einschalten des C 128 sind die Funktionstasten bereits mit nützlichen Anweisungen belegt.

Programme vom C 64 auf den C 128

Wie kann ein Datenverwaltungsprogramm vom C 64 auf den C 128 umgesetzt werden? Meine Anfragen bei Computershops variierten zwischen »Das ist völlig unmöglich« und »Da gibt es gar keine Schwierigkeiten«.

Hans Merkert

Für die Umsetzung der Programme ist einiges zu beachten. Zu allererst sollten in den C 64-Programmen keine Maschinenroutinen enthalten sein, da der C 128 mit einer völlig anderen Speicherverwaltung arbeitet. Ist die Dateiverwaltung komplett in Basic geschrieben, können Sie das Programm ohne Probleme auf dem C 128 laufen lassen. Basic 2.0 ist aufwärtskompatibel zum Basic 7.0. Natürlich bietet Basic 7.0 viel mehr Möglichkeiten, auch in Hinsicht auf Dateiverwaltung. Die einzelnen Befehle vor allem zum Be- und Verarbeiten von relativen Dateien sind weitaus komfortabler als die Programmierung im C 64-Modus. Die umständliche Ansteuerung der einzelnen Datensätze mit wüsten »CHR$«-Kombinationen ist durch einen komfortablen »RECORD«-Befehl ersetzt. Auch die Handhabung sequentieller Dateien gestaltet sich durch das komfortable Basic 7.0 wesentlich einfacher. Durch die doppelte Speicherkapazität des C 128 können natürlich auch größere Datenmengen im Speicher verarbeitet werden.

Speichererweiterung und Maus

Wann sind endlich Maus und Speichererweiterung für den C128 erhältlich und was kosten sie?

Achim Pankalla

Angekündigt sind diese Systemkomponenten bekanntlich schon seit der Auslieferung des C 128. Doch leider hat eine Ankündigung nichts mit einer Auslieferung zu tun. Mit den Speichererweiterungen 1700 (256 KByte) und 1750 (512 KByte) dürfte nach Aussagen von Commodore zum Weihnachtsgeschäft 1986 zu rechnen sein. In den USA werden diese bereits angeboten. Die Maus dürfte noch etwas länger auf sich warten lassen.

Cursor-Positionierung

Mit welcher Anweisung kann ich beim C128 den Cursor an eine beliebige Bildschirmstelle positionieren? Benötige ich dazu einen POKE-Befehl?

Jörg Welders

Um den Cursor zu positionieren, bietet der C 128 eine verblüffend einfache Lösung an. Die Positionierung erfolgt über den Grafikbefehl »CHAR«. Wollen Sie den Cursor zum Beispiel etwa in der Bildschirmmitte stehen haben, geben Sie folgenden Befehl ein: »CHAR ,38,12«, wobei 38 die Spalte und 15 die Zeile angibt.

Software für den C 128

Gibt es bereits Software für den C128? Wenn ja, welche und was kostet Sie?

Jens Bildner

Da der C 128 drei Betriebssysteme hat (C64, C128 und CP/M), können Sie auf ein reichhaltiges Softwareangebot zurückgreifen. Für den C 64 gibt es jede Menge Spiele und auch ernsthafte Anwendungen. Im CP/M-Modus steht eine riesige Palette von Branchen- und allgemeiner Anwendungssoftware zur Verfügung. Sie erhalten praktisch alles von der Textverarbeitung über Verwaltungssysteme bis hin zu den verschiedensten Compilern. Besonders bieten sich CP/M-Klassiker wie dBase II, Wordstar, Multiplan und MBasic an, um nur eine kleine Auswahl zu nennen. Für den C 128-Modus sind ebenfalls einige leistungsfähige Programme erhältlich. Es gibt mehrere Textverarbeitungen und auch einige Compiler. Beachten Sie dazu bitte die entsprechenden Testberichte und Anzeigen in der Zeitschrift 64'er. Ein neuer Computer hat immer mit dem Problem der Verzögerung des Softwareangebots nach seiner Einführung zu kämpfen. Dies trifft auch für den C 128 in seinem ureigensten Modus zu. Programm-Neuentwicklungen und Adaptionen bestehender Programme für den C 64 auf dem C 128 beweisen, daß sich mit diesen nahezu professionell arbeiten läßt. Neuerdings stehen auch eine leistungsfähige Datenbank und ein Kalkulationsprogramm zur Verfügung.

Der C 128 wächst...

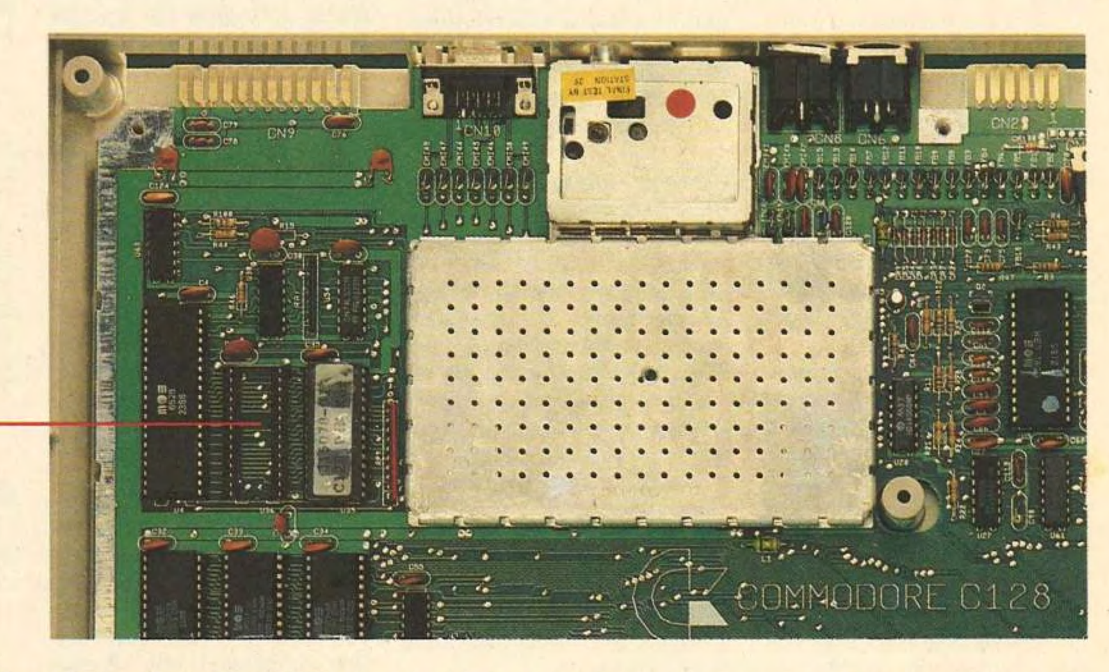


Bild 1. Die Platine des C128. Der freie Steckplatz neben dem ROM des C128-Betriebssystems kann leicht für eigene Programme in EPROMs genutzt werden.

Wenn Sie Ihren C128 schon einmal geöffnet haben, dann wird Ihnen sicherlich der leere Sockel auf der Platine aufgefallen sein. Was man damit alles anfangen kann, sagt Ihnen der folgende Artikel.

Neben einer Vielzahl an Bausteinen enthält der C128 auch einen Sockel, der beim Kauf des Geräts noch nicht belegt ist. Dieser freie Steckplatz eröffnet uns eine Vielzahl an Möglichkeiten. Schrauben Sie das Gehäuse Ihres C128 auf (Vorsicht! Garantieverlust), dann erkennen Sie den leeren Sockel auf der linken Platinenseite neben dem ROM des C128-Betriebssystems (Bild 1). Der Sockel trägt die Bezeichnung U 36 und ist für ein 16- oder 32 KByte EPROM (27128 oder 27256) vorgesehen.

Was kann man nun mit diesem Zusatz anfangen? Leider gibt es bisher so gut wie keine Information über die Beschaltung und das Ansprechen eines EPROMS in diesem Sockel, so daß wir für Sie einmal auf Entdeckungsreise gegangen sind.

Wie Sie sicherlich wissen, kann man sowohl in den C64 als auch in den C128 Erweiterungsmodule in einen dafür vorgesehenen Steckplatz auf der Rückseite des Computers stecken. Diese Buchse nennt sich Erweiterungs- oder Expansion-Port.

Der C128 kann sowohl die Module eines C64 als auch seine »eigenen« erkennen und schaltet dabei in den entsprechenden Modus. Die Module des C64 werden dabei dadurch identifiziert, daß sie die EXROM- und die GAME-Leitung für ihre Zwecke verwenden. Sie schalten sich also hardwaremäßig ein.

Beim C128 gingen die Entwickler einen anderen Weg. Hier kommt die MMU (»memory management unit« oder »Speicherverwaltungs-Einheit«) ins Spiel. Da der C128 mehrere Speicherbänke zu je 64 KByte verwalten kann, muß zwischen den einzelnen Bänken auch umgeschaltet werden, um auf jedes Byte des Speichers zugreifen zu können.

Diese Umschaltung regelt die MMU, wobei auf Wunsch eine bestimmte Konfiguration des Speichers eingestellt werden kann. Dazu muß man wissen, daß der Speicher des C128 in einen RAM-, einen unteren, einen mittleren und einen oberen Speicherbereich aufgeteilt ist. Dabei geht der untere Bereich von $4000 bis $7FFF, der mittlere von $8000 bis $BFFF und der obere von $C000 bis $FFFF. Der RAM-Bereich von $0000 bis $3FFF kann, wie der Name schon sagt, nur RAM enthalten. Die drei anderen Bereiche können jedoch wahlweise mit einer bestimmten Belegung versehen werden (Bild 2).

Wir wollen die Ausführungen an dieser Stelle zwar so knapp wie nur möglich halten, die wichtigsten Hinweise zur Einstellung der Speicherkonfiguration sollen jedoch an dieser Stelle gegeben werden. Für detailliertere Informationen kann das Handbuch zum C128 zu Rate gezogen werden.

Für die Einstellung der Speicherkonfiguration existiert in der MMU ein wichtiges Register: das Konfigurationsregister, dessen Belegung in Bild 3 dargestellt ist.
Die Bitbelegung im einzelnen:

Bit sieben und sechs geben die Nummer der aktuellen RAM-Bank an, die für die gesamten 64 KByte gilt, die gerade bearbeitet werden. Da in der Grundversion des C128 nur die Bänke Null und Eins existieren, können wir Bit sieben vernachlässigen. Bit sechs gibt also die Banknummer an.

Der RAM-Bereich von $0000 bis $3FFF ist, wie schon gesagt, immer mit RAM belegt. Dabei ist die Banknummer in Bit sechs ausschlaggebend.

Bei den Bits fünf und vier haben Sie vier verschiedene Möglichkeiten. Sind beide gelöscht, so ist im oberen Bereich der aktuellen Bank von $C000 bis $FFFF das Betriebssystem-ROM (Kernel) des C128 eingeblendet. Es steckt auf der Platine an Position U 35 und enthält 16 KByte Speicher. Sind die Bits fünf und vier beide gesetzt, so adressiert die MMU im betreffenden Speicherbereich 16 KByte RAM der entsprechenden Speicherbank.

So, und jetzt wird es interessant. Sie haben nämlich jetzt noch zwei Bitkombinationen übrig, die die Bits vier und fünf annehmen können. Ist Bit fünf gesetzt und Bit vier gelöscht, so wird der Expansion-Port des C128 adressiert. Steckt hierin nun ein Modul, so wird es in den Speicherbereich $C000 bis $FFFF eingeblendet und behandelt, als wäre es ein ganz »normales« Kernel.

Ist der Zustand der Bits fünf und vier hingegen umgekehrt, also Bit fünf gelöscht und Bit vier gesetzt, so wird der interne Zusatzsteckplatz U 36, der uns im folgenden noch genauer interessieren soll, adressiert. Hier kann nun zum Beispiel ein völlig neues Betriebssystem integriert sein.

Wie Sie sehen, haben im C 128 sowohl ROM und RAM als auch Module und ROMs im Zusatzsteckplatz die gleiche Priorität. Es wird lediglich im Konfigurationsregister das eingestellt, was der Anwender gerne als Kombination haben möchte.

Die Bits drei und zwei haben genau die gleiche Funktion wie die Bits fünf und vier. Hier wird lediglich der Adreßbereich von $8000 bis $BFFF geschaltet, so daß Sie weitere Kombinationsmöglichkeiten haben. Es steht Ihnen also zum Beispiel frei, von $8000 bis $BFFF ein Modul einzuschalten, während der Bereich $C000 bis $FFFF RAM enthält. Der Speicherbereich von $8000 bis $BFFF wird übrigens normalerweise durch ein ROM repräsentiert, das einen Teil des Basic-Interpreters und den TEDMON des C 128 enthält. Es ist auf der Platine im Sockel U 34 untergebracht.

Bit eins ist für den Adreßbereich $4000 bis $7FFF zuständig. Hier können Sie kein Modul oder externes ROM mehr einblenden, sondern haben nur noch die Wahl zwischen Basic-ROM oder RAM. Bei dem Basic-ROM handelt es sich um den ersten Teil des Interpreters, der in einem 16 KByte ROM in dem Steckplatz U 33 enthalten ist.

Bit null des Konfigurationsregisters hat eine Sonderfunktion, die den Adreßbereich $C000 bis $FFFF betrifft. Im Bereich von $D000 bis $DFFF kann nicht nur RAM oder ROM eingeblendet werden; hier liegen vielmehr auch noch die I/O-Bausteine des C 128. Soll also ein Zugriff auf VIC, SID, CIA oder andere Peripherie erfolgen, so muß von $D000 bis $DFFF der Ein-/Ausgabebereich des Computers eingeblendet werden. Ist Bit null gelöscht, so ist der I/O-Bereich eingeblendet, wobei sich gleichzeitig der Speicherbereich infolge der ausgefallenen 4 KByte auf 12 KByte vermindert ($C000 bis $CFFF und $E000 bis $FFFF). Ist das Bit gleich eins, so kann auf den zusätzlichen Speicherbereich zugegriffen werden.

Schön und gut, werden Sie jetzt sagen. Man kann also auf das ROM im Zusatzsteckplatz (das wir übrigens ab jetzt als Funktions-ROM bezeichnen) zugreifen. Toll, aber was bringt mir das?

Auf das Funktions-ROM können Sie nicht nur zugreifen, es wird vielmehr vom Betriebssystem des C 128 unterstützt. Das »Wie« soll uns dabei nun interessieren.

Wie bereits erwähnt, werden Module beim C 128 nicht hardwaremäßig initialisiert und gestartet. Der C 128 regelt das Einschalten eines Moduls sehr viel eleganter.

Wird ein Reset ausgeführt, so blendet das Betriebssystem des Computers mit Hilfe des Konfigurationsregisters nacheinander vier verschiedene Erweiterungseinheiten in die aktuelle Bank ein: Funktions-ROM von $8000 bis $BFFF, Funktions-ROM von $C000 bis $FFFF, Expansion-Port von $8000 bis $BFFF und Expansion-Port von $C000 bis $FFFF. Es wird also jetzt beispielsweise das Funktions-ROM bei $8000 eingeblendet. Dann liest das Betriebssystem die Speicherstellen $8009, $800A und $800B aus und schaut nach, ob dort die Kennung »CBM« enthalten ist. Ist das nicht der Fall, so wird der Adreßbereich $C009 bis $C00B untersucht. Findet der C 128 auch hier nichts, prüft er noch die gleichen Adreßräume für den Expansion-Port und führt bei

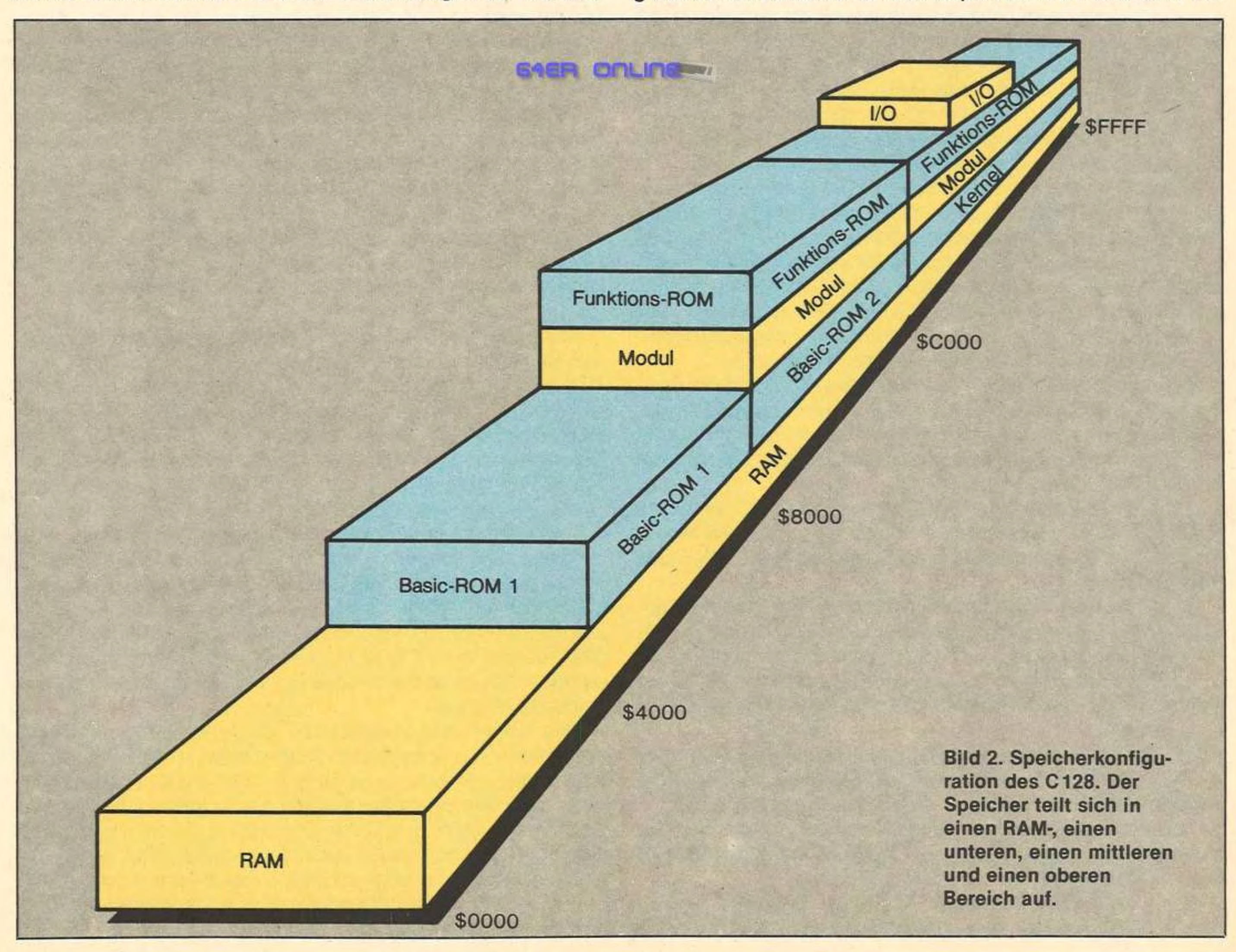


Bild 2. Speicherkonfiguration des C 128. Der Speicher teilt sich in einen RAM-, einen unteren, einen mittleren und einen oberen Bereich auf.

Bit	Erklärung
7	höherwertiges Bit der Nummer der Speicherbank
6	niederwertiges Bit der Nummer der Speicherbank
5/4	Adreßbereich $C000 bis $FFFF:
	0/0 – Kernel eingeblendet
	0/1 – Funktions-ROM eingeblendet
	1/0 – Steckmodul im Expansion-Port eingeblendet
	1/1 – RAM eingeblendet
3/2	Adreßbereich $8000 bis $BFFF:
	0/0 – Basic-ROM 2 eingeblendet
	0/1 – Funktions-ROM eingeblendet
	1/0 – Steckmodul im Expansion-Port eingeblendet
	1/1 – RAM eingeblendet
1	Adreßbereich $4000 bis $7FFF:
	0 – Basic-ROM 1 eingeblendet
	1 – RAM eingeblendet
0	Adreßbereich $D000 bis $DFFF (I/O):
	0 – I/O-Bausteine eingeblendet
	1 – Konfiguration von Bit 4 und 5 eingeblendet

Bild 3. Die Belegung des Konfigurationsregisters in der MMU des C 128. Die Grundadresse ist $D500 beziehungsweise $FF00.

einem Mißerfolg die »normale« Reset-Routine aus, um schließlich das Basic 7.0 zu starten.

Wir haben also auch beim C 128 die Abfrage auf eine CBM-Kennnung, die wir schon vom C 64 her kennen.

Wie beim C 64, so erfolgt auch beim C 128 ein Start des entsprechenden Moduls, wenn die Bedingungen dafür erfüllt sind. Hier sind jedoch noch ein paar Einzelheiten zu beachten.

Beim C 64 wird im Falle eines Modulstarts indirekt an die Adresse gesprungen, die in $8000 und $8001 gespeichert ist. Der C 128 springt hingegen direkt in das Modul oder das Funktions-ROM. Wurde also die Kennung festgestellt, so erfolgt der Sprung an die Grundadresse, bei der das Modul oder ROM gefunden wurde. Diese Grundadresse ist je nach gefundener Kennung entweder $8000 oder $C000.

Theoretisch kann mit dem Befehl RTS in das Betriebssystem des C 128 zurückgesprungen werden, da ein JSR in das Funktions-ROM stattgefunden hat. Es ist hierbei jedoch zu beachten, daß das Funktions-ROM anders adressiert wird als das Kernel, so daß das Kernel unter Umständen ausgeblendet wird, um das Funktions-ROM einzuschalten. Der C 128 muß also immer mit sogenannten »long-jumps« arbeiten, was einem Umweg über das RAM entspricht.

Neben der CBM-Kennung im Funktions-ROM ist aber auch noch ein anderer Faktor für einen Sprung in das ROM ausschlaggebend. Es handelt sich hierbei um ein Byte, das immer genau vor der CBM-Kennung zu finden ist, also bei $8008, beziehungsweise $C008.

Steht dieses Byte auf $00, so wird das eingesteckte Funktions-ROM oder Modul nicht als Autostart-Modul erkannt, und es erfolgt kein Sprung in das eingesteckte ROM oder EPROM.

Steht das Byte hingegen auf $01, so wird das Funktions-ROM oder das eingesteckte Modul direkt in der Reset-Routine angesprungen und zwar, bevor der gesamte Computer in einen definierten Zustand versetzt worden ist. Hier hat man also sehr früh die Möglichkeit, in den Betrieb des C 128 einzugreifen.

Steht das Byte in $8008 ($C008) hingegen auf $02 oder einem beliebigen anderen Wert bis $FF, so geschieht die dritte Möglichkeit der Ansteuerung des Funktions-ROM.

Das interessante an dem ROM ist nämlich, daß bei Vorhandensein der Kennung auch an späterer Stelle der Reset-Routine noch ein Einsprung erfolgen kann.

Wenn Sie Ihren C 128 einschalten und das Diskettenlaufwerk schon vorher eingeschaltet war, so erfolgt die Meldung des Basic 7.0 auf dem Bildschirm. Bevor jedoch das »READY.« mit dem Cursor erscheint, erfolgt ein Diskettenzugriff, der eine bootfähige Diskette erkennen und gegebenenfalls ein Programm laden und automatisch starten soll.

Betrachtet man sich die Routinen des Betriebssystems jedoch genauer, so kann man erkennen, daß vor dem Diskettenzugriff noch einmal die vier möglichen ROM-Variationen abgefragt werden, so wie es direkt nach dem Einschalten des Computers passiert.

Hier kann der Programmierer nun ansetzen, wenn er das Steuerbyte in $8008 ($C008) mit einem Wert zwischen $02 und $FF versieht. Wird dieser Wert erkannt, so startet der C 128 das Zusatz-ROM vor dem Zugriff auf die Diskette, aber nachdem bereits das gesamte C 128-Computersystem initialisiert wurde.

Sie haben als Anwender also drei Möglichkeiten: Entweder Sie programmieren sich ein eigenes Betriebssystem, das eine komplett eigene Reset-Routine und Initialisierung des Computers enthält. Dann ist es sinnvoll, wenn Sie direkt am Anfang der Reset-Routine in das eigene System springen (Byte also auf $01 setzen). Wollen Sie jedoch kein Betriebssystem entwickeln, sondern nur eigene Maschinenprogramme in ein Modul einbinden, so ist es besser, wenn Sie Ihr Programm erst dann starten lassen, wenn sich das gesamte Computersystem bereits in einem definierten Zustand befindet (Byte-Wert $02 bis $FF). Es ist jedoch auch möglich, daß Sie Software besitzen, bei der es angebracht ist, einen Teil in ein Modul und einen Teil auf Diskette zu haben. Dann wäre es am günstigsten, wenn Sie den Wert $00 in das Steuerbyte schreiben und das Modul mit Ihrem eigenen Programm von der Diskette aktiv einschalten und anspringen.

Ein kleiner Fehler im Betriebssystem des C 128 wäre vielleicht noch erwähnenswert: Wenn Sie die Kennung $01 im Steuerbyte stehen haben und aus der frühen Initialisierung mit RTS wieder in die Reset-Routine des C 128 zurückspringen, so erfolgt der Aufruf des Moduls noch einmal beim Basic-Start. Sie bekommen also zwei Autostarts Ihres Moduls, da der zweite Start nur noch auf das Steuerbyte ungleich Null abfragt und nicht verlangt, daß ein Wert größer gleich $02 vorhanden ist. Dieser Fehler wird uns aber in der Regel nicht stören, da ein Rücksprung aus dem Modul in das Betriebssystem in den wenigsten Anwendungsfällen sinnvoll ist.

Wenn Sie ein EPROM mit 16 KByte Speicherkapazität in den Sockel U 36 einstecken, haben Sie natürlich im Bereich ab $8000 und ab $C000 den gleichen Inhalt bei der Initialisierung. Sie können sich also hier auswählen, ob Sie den $8000 bis $BFFF- oder den $C000 bis $FFFF-Bereich als Modulbereich anwählen. In den meisten Fällen wird $8000 bis $BFFF der günstigere Bereich sein, da man dann das Betriebssystem des Computers ab $C000 einblenden und aus dem Modul heraus sehr einfach anspringen kann.

Haben Sie hingegen ein 32-KByte-EPROM im Sockel stecken, so muß natürlich der gesamte Bereich von $8000 bis $FFFF auf Funktions-ROM geschaltet werden, um einen Zugriff zu ermöglichen. Ob man dann die Startadresse bei $8000 oder bei $C000 liegen hat, ist nur eine Frage des eigenen Geschmacks und hat für den Betrieb des EPROMs keine Bedeutung.

Eine mächtige Angelegenheit also, das Funktions-ROM. Sie können von der eigenen Programmiersprache bis hin zu eingebauten Kopier- oder DFÜ-Systemen alles realisieren, was Sie wollen. Und das alles steht auf Wunsch, direkt nach dem Einschalten des Computers, ohne einen Verlust des Original-Betriebssystems zur Verfügung. Versuchen Sie es doch einmal, und schicken Sie uns Ihre eigenen Betriebssysteme für den C 128. Wir sind gespannt auf die ersten Programme für den C 128, die keine RAMs mehr benötigen. (ks)

IEEE-Bus für den C 128

Wollen Sie professionelle Peripheriegeräte an Ihren Computer anschließen? Hier finden Sie die notwendige Bauanleitung dafür. Zusätzlich werden eine ganze Menge Extras geboten.

Wozu braucht man überhaupt einen IEEE-488-Bus am C128? Nun, die Antwort ist sehr einfach. Über ein paralleles IEEE-Bus-Modul können Sie die Drucker und die Diskettenlaufwerke mit erheblich höherer Speicherkapazität aus der Commodore 4000er- und 8000er-Serie anschließen. Interessant ist vor allem das Einzellaufwerk SFD 1001 mit einer Speicherkapazität von 1 MByte und die Doppelfloppy CBM 8250 LP mit 2 x 1 MByte Speicherkapazität. Besonders bei den Laufwerken findet man häufig günstige Angebote (siehe Anzeigenteil).

Schnellere Floppy

Besonders das Floppylaufwerk SFD 1001 weist etliche Vorzüge gegenüber den Laufwerken 1571 und 1541 auf. Im einzelnen sind diese:

1. Das Laufwerk 1571 emuliert im C 64-Modus des C 128 die Floppystation 1541 - zumindest, was die Geschwindigkeit angeht. Eine SFD 1001 ist beim Lesen aber fast sechsmal schneller, beim Schreiben etwa dreimal so schnell wie die 1541 und doppelt so schnell wie die 1571 im C128-Modus. Auch sequentielle und relative Dateien werden entsprechend schneller bearbeitet. Angenehm bemerkbar macht sich die Geschwindigkeitssteigerung auch im CP/M-Betrieb, bei dem häufig auf Diskette zugegriffen wird.
2. Die Speicherkapazität der SFD 1001 ist gegenüber der 1541 sechsmal größer. Gegenüber der 1571 beträgt sie immer noch das dreifache. Es treten auch keine Probleme mit den Diskettenformaten 1541/einseitig und 1571/doppelseitig mehr auf.

Wenn Sie eine 1541 am C128 betreiben, können Sie diese mit Hilfe der in Ausgabe 1/86 vorgestellten IEEE-Platine auf den parallelen Übertragungsstandard umrüsten. Die Vorteile der höheren Übertragungsge schwindigkeit können Sie im C64-Modus, im C 128-Modus und unter CP/M nutzen.

Ein Modul und zwei EPROMs

Um den IEEE-488-Bus an Ihrem C 128 betreiben zu können, benötigen Sie ein Hardware-Modul (Bild 1) und zwei Betriebssystem-EPROMs.

Das eine EPROM des Typs 27128 beinhaltet das Basic V2 und das modifizierte Betriebssystem für den C 64-Modus. Auf der Programmservice-Diskette zu dieser Ausgabe ist das zum Brennen des EPROMs notwendige Programm enthalten. Sie finden es dort unter dem Namen »IEEE KERN 64«. Im Bild 2 sehen Sie den Platinenausschnitt mit den beiden Betriebssystem-ROMs. Dieses EPROM können Sie direkt an Stelle des mit »2« gekennzeichneten ROMs einsetzen. Ein Adaptersockel ist dabei nicht notwendig.

Das zweite EPROM, ebenfalls ein 27128-Typ mit einer Zugriffszeile von 250 ns, ersetzt das mit »1« gekennzeichnete ROM (Bild 2). Es beinhaltet das neue Betriebssystem für den C 128-Modus. Das notwendige Programm finden Sie ebenfalls auf der Programmservice-Diskette unter »IEEE KERN 128«.

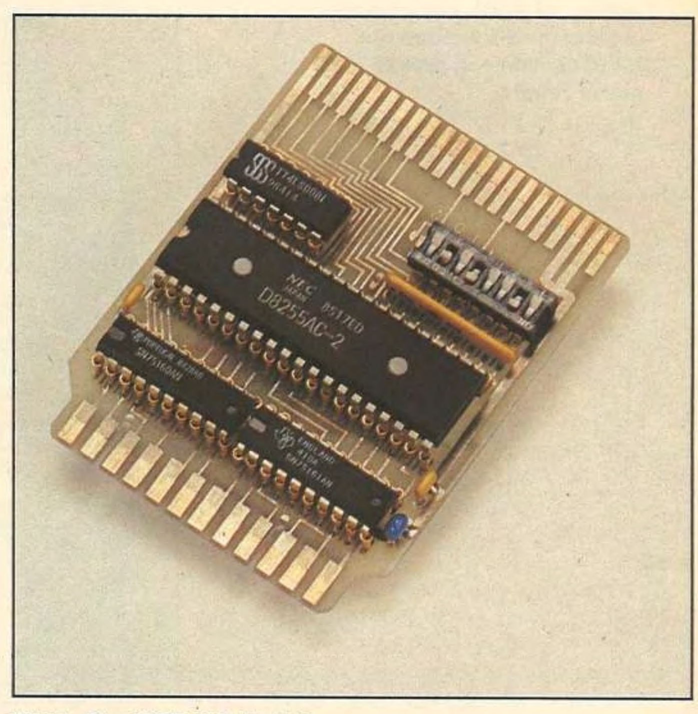


Bild 1. Das IEEE-488-Modul

Die beiden EPROMs für die Betriebssysteme sind auf der Stückliste (Tabelle 1) nicht mit aufgeführt. Sie enthält nur die Bauteile für das Hardware-Modul. Auf dieses Modul wollen wir jetzt auch näher eingehen.

Die Schaltung

Für eine parallele Datenübertragung nach der IEEE-Norm benötigen Sie eine kleine Platine für den Expansion-Port (Bild 1). Die Schaltung finden Sie im Bild 3.

Der wichtigste Baustein (IC2) ist ein »Programmable Peripheral Interface«, abgekürzt PPI. Dieser Baustein des Typs µPD8255AC kann ohne weiteres mit 2 Megahertz betrieben werden. Geschwindigkeitsprobleme wird es also keine geben. Da IC2 jedoch nicht zur Familie der 65xx-Bausteine gehört, ist eine Anpassung der Steuersignale notwendig. Dies erfolgt über die vier Gatter des 74LS00 (IC1).

Die beiden ICs SN75160 (IC3) und SN75161 (IC4) sind Treiber-Bausteine mit integriertem Widerstandsnetzwerk, wie es nach der IEEE-Norm verlangt wird.

Mit dem 8fach DIL-Schalter wird die Schnittstelle auf Ihre Gerätekonfiguration angepaßt.

Kommen wir aber nochmals zum I/O-Baustein µPD8255. Er wird über den für die Erweiterungen I/O 1 vorgesehenen Adreßbereich $DE00 bis $DEFF angesprochen. Sie können das Modul aber auch auf den Erweiterungsbereich I/O 2 ($DF00 bis $DFFF) verlegen, indem Sie die Brückenverbindung rechts neben den DIP-Schaltern (Bild 4) auftrennen und umlöten.

Von dem neuen Betriebssystem wird allerdings der Bereich I/O 1 angesprochen. Die Software spricht die Register von $DE40 bis $DE43 für Read-Funktionen und die Register von $DE48 bis $DE4B für Write-Funktionen an.

Der µPD8255 besitzt insgesamt drei 8 Bit breite Ports (PA, PB, PC). Im Bild 5 ist der µPD8255 nochmals mit allen Pin-Belegungen abgebildet. Welche der I/O-Anschlüsse als Da-

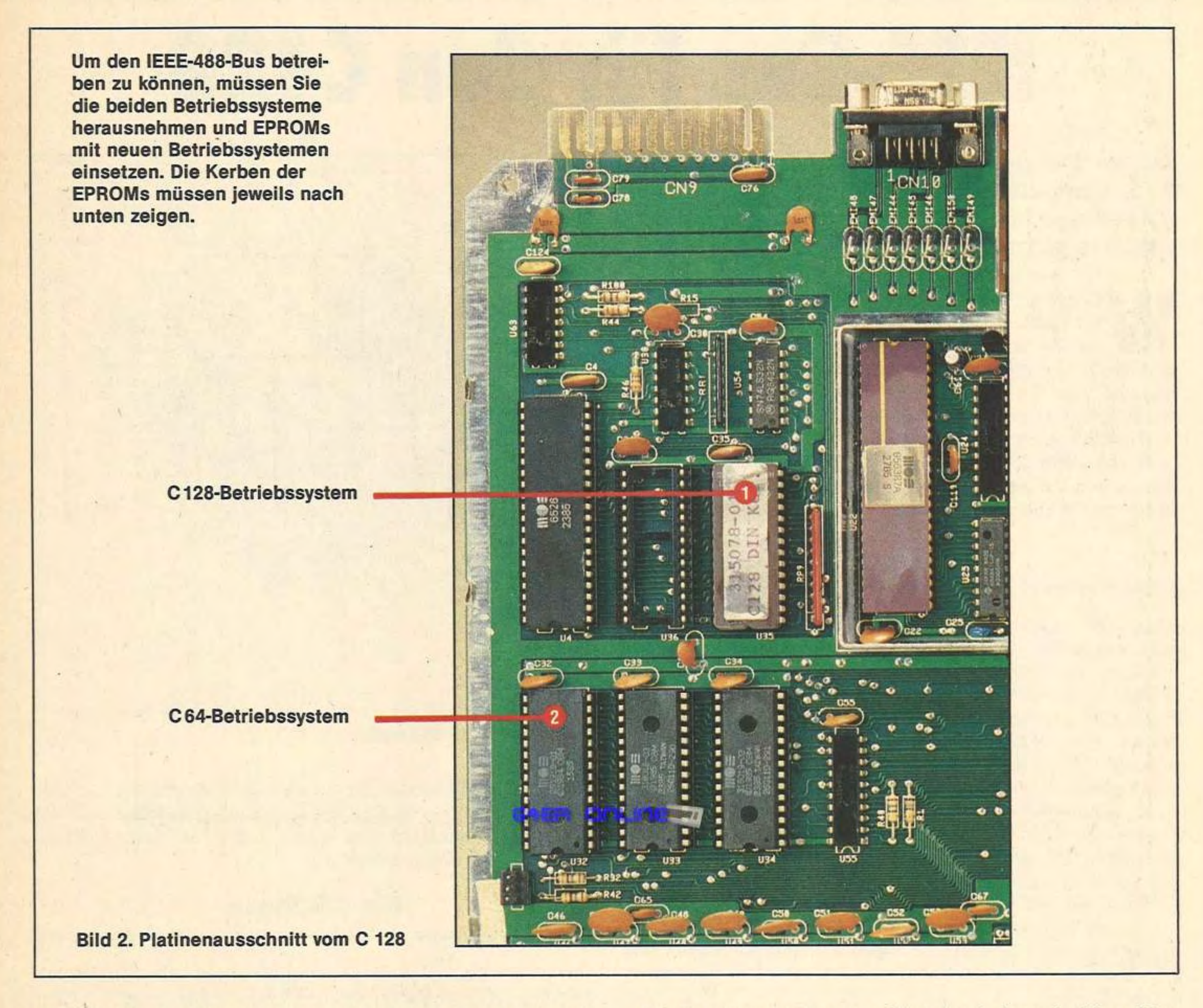


Bild 2. Platinenausschnitt vom C 128

ten- oder Steuerleitungen verwendet werden und welche mit den DIP-Schaltern geschaltet werden, das finden Sie in der Tabelle 2.

Mit den Leitungen TE und DC der Bus-Treiber (IC3, IC4) wird die Datenrichtung und der Zustand des Computers (Controller/Talker/Listener) festgelegt.

Tips für den Aufbau

Um die Schaltung aufzubauen, brauchen Sie eine doppelseitige Platine. Die Platinenlayouts für beide Seiten finden Sie in den Bildern 6 und 7. Die Abmessungen der Platine betragen 59 x 75 mm. Für die Herstellung doppelseitiger Platinen empfiehlt es sich, Folien vom Layout herzustellen, die beiden Folien genau übereinander zu legen und mit Tesa-Film zu einer Art Tasche zusammenzukleben. Stecken Sie dann eine Platine mit Fotoschichten in diese Tasche, lassen Sie diese auf der offenen Taschenseite überstehen und kleben Sie die Platine mit Tesa-Film an der Tasche fest. Nun können Sie die Platine auf beiden Seiten belichten.

An der IEEE-Bus-Seite muß die Platine später auf die Maße des 24poligen Platinensteckers zurechtgesägt werden. Achten Sie auch auf die notwendigen Schlitze. Sie liegen anders als am User-Port! Im Layout (Bestückungsseite) sind kleine, halbkreisförmige Markierungen vorhanden, wo die Schlitze gesägt werden müssen.

Die Durchkontaktierung erfolgt durch doppelseitiges Verlöten der IC-Sockel. Gedrehte Sockelbeinchen ohne Kunststoffrahmen (Carrier-Fassungen), wie sie auf dem Bild 1 zu sehen sind, erleichtern das Löten auf der Bestückungsseite. Sie werden mit einem »Blech-IC« als Träger verkauft. Weitere Durchkontaktierungen sind nicht notwendig.

So schließen Sie Geräte an

An der Platine sind zwei Anschlüsse vorhanden. Mit der 44poligen Seite wird die Platine direkt in den Expansion-Port gesteckt. Die Bestückungsseite zeigt dabei nach oben. Den Commodore-IEEE-488-Bus bildet der 24polige Anschluß mit den beiden Schlitzen (siehe Bild 1). Über ein Commodore-IEEE-488-Bus-Kabel können Sie die Peripheriegeräte der Serien 4000 und 8000 anschließen. Eine Gegenüberstellung der Pinbelegung des Commodore-IEEE-488-Anschlusses und des 24poligen IEEE-488-Normsteckers finden Sie in der Tabelle 3.

Daten seriell und parallel

Der Datenverkehr mit den Peripheriegeräten kann seriell und parallel erfolgen. Unter der gleichen Gerätenummer kann ein

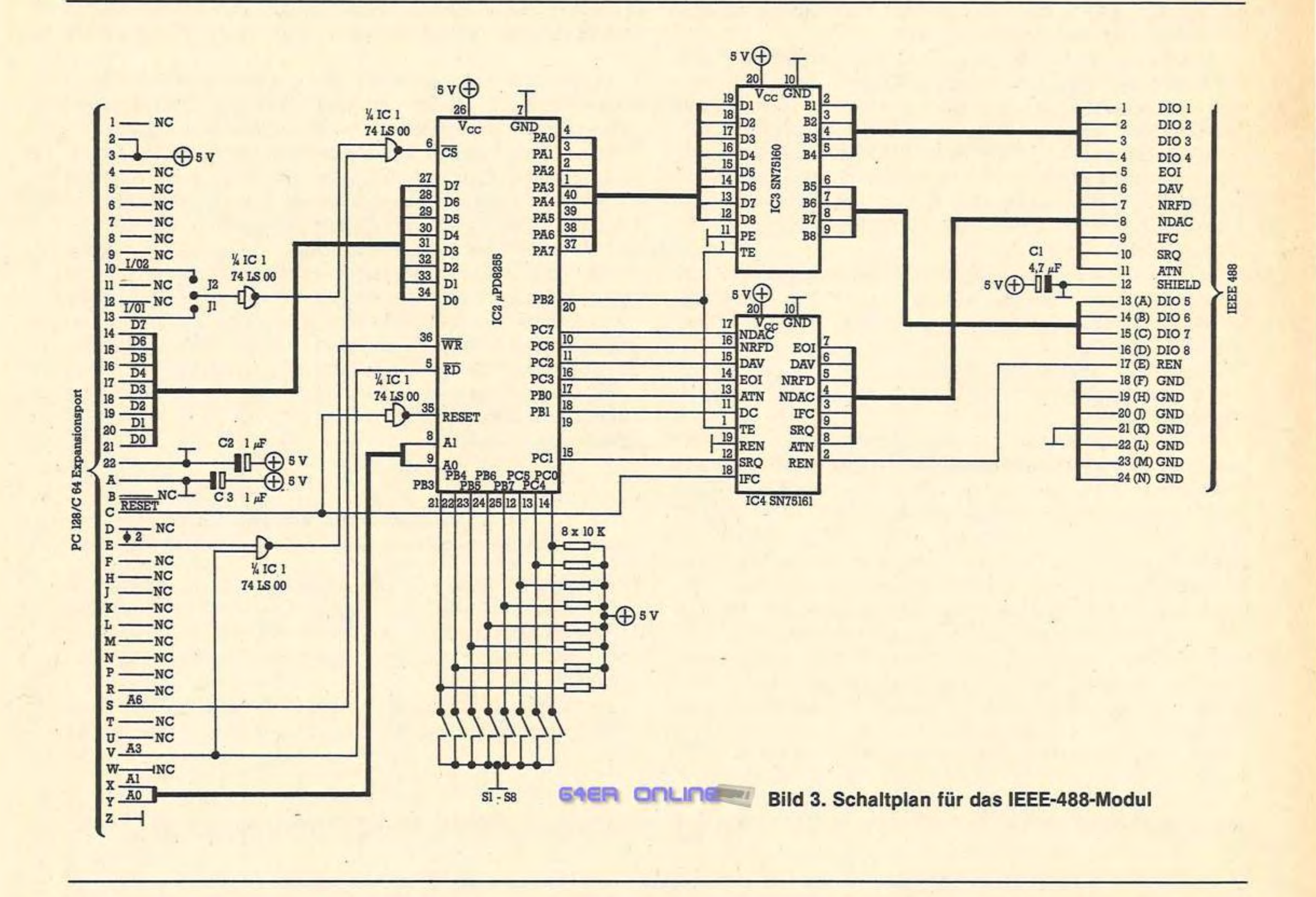


Bild 3. Schaltplan für das IEEE-488-Modul

Gerät seriell angesprochen werden und ein zweites parallel. Auf diese Weise ist es möglich, zwei Floppylaufwerke zu betreiben, ohne die Gerätenummer zu ändern.

Die Art des Zugriffs (seriell oder parallel) können Sie entweder softwaremäßig über ein Register oder mit den DIP-Schaltern bestimmen. Mit dem DIP-Schalter 8 wird eine der beiden Varianten vorgegeben. Ein Wechsel der Zugriffsart ist auch während dem Betrieb möglich.

Die Übersicht, was mit welchem Schalter eingestellt wird, finden Sie in der Tabelle 4 unter »Switch«. In der Mitte ist eine gängige Schalterkombination abgebildet. Die Bedeutung der einzelnen Schalter wollen wir Ihnen aber auch noch genauer erklären.

Schalter 1 invertiert die Funktion der Taste <CBM> nach dem Einschalten und nach einem Reset. Setzen Sie Schalter 1 auf »ON«, so gelangen Sie nach dem Einschalten in den C 64-Modus, ohne die Taste <CBM> zu drücken. Haben Sie diese Taste gedrückt, meldet sich der Computer im C 128-Modus.

Schalter 2 legt die Farben des 40-Zeichen-Bildschirms fest. Im C 128-Modus können Sie zwischen schwarz/weiß und blau/weiß wählen. Im C 64-Modus stehen die Kombinationen schwarz/grau oder blau/grau zur Verfügung.

Ist Schalter 3 auf »ON« gestellt, erfolgt die Datenausgabe für die Gerätenummer 4 (Drucker) über eine Centronics-Schnittstelle am User-Port. Die FLAG-Leitung wird dabei als »Acknowledge« und die PA2-Leitung als »Strobe« verwendet. Wenn Sie im OPEN-Befehl die Sekundäradresse 0 wählen, werden alle Zeichen im »CBM-ASCII-Code« ohne eine Codewandlung ausgegeben. Mit der Sekundäradresse 7 wird eine Wandlung der Klein-/Großschrift durchgeführt. Die Codewandlung erfolgt bis zum Zeichen CHR$(95).

Steht der Schalter 3 auf »OFF«, so wird über die Gerätenummer 4 nicht der User-Port angesprochen, sondern der serielle oder parallele Bus.

Die Schalter 4 bis 7 bestimmen die Art des Zugriffs für die Gerätenummern 4, 5, 8 und 9. Dabei bedeutet »OFF« seriell und »ON« parallel.

Der Schalter 8 trägt die Bezeichnung »Request Switches?« (Schalter abfragen?). Im Zustand »ON« werden die übrigen Schalter im Betrieb abgefragt. Im Zustand »OFF« werden die entsprechenden Bits des Registers $DD0C abgefragt.

Einstellung per Software

Um eine softwaremäßige Einstellung zu erreichen, wird das serielle Schieberegister ($DD0C) des CIA-Bausteins 2 benutzt. Die Bedeutung der einzelnen Bits finden Sie als Übersicht ebenfalls in der Tabelle 4. In der Mitte dieses Tabellenabschnittes ist wiederum die gängigste Bit-Kombination angegeben. Wie bei den Schaltern wollen wir Ihnen auch hier die einzelnen Bits noch einmal genau erklären.

Die Bits 0, 1 und 2 bestimmen die Art des Zugriffs für die Gerätenummern 4, 8 und 9. Bei gelöschtem Bit wird der serielle Bus, bei gesetztem Bit der parallele Bus verwendet.

Bit 3 bestimmt, ob die Gerätenummer 4, die Centronics-Schnittstelle, angesprochen wird. Bei gesetztem Bit läuft die Datenausgabe über den User-Port (Centronics). Bei

gelöschtem Bit 3 entscheidet das Bit 0 über die Art der Ausgabe. Bit 3 hat also Vorrang vor Bit 0.

Mit Bit 4 können Sie das Modul als »nicht vorhanden« (Bus not available) schalten. Dazu braucht nur Bit 4 gelöscht werden. Im Normalfall ist das Bit bei eingestecktem Modul gesetzt. Nach dem Einschalten wird dieses Bit überprüft, bevor die Schalter abgefragt werden. Es hat also die höchste Priorität. Fehlt das Modul nach dem Einschalten, so werden die Bits 0 bis 3 vom Betriebssystem gelöscht. Bit 4 kann auch benutzt werden, um eine Schalterabfrage softwaremäßig zu unterdrücken.

Die Bits 5 bis 7 dürfen Sie nicht benutzen, da sie vom Betriebssystem verwendet werden. Aus diesem Grund sollten Sie die einzelnen Bits (0 bis 4) von Basic aus mit folgendem Befehl setzen:

POKE56588, (PEEK(56588) OR X)

Anstelle X werden je nach Bit die Werte 1, 2, 4, 8 oder 16 eingesetzt.

Die Bedeutung von Bit 5 wollen wir Ihnen aber auch erläutern. Das Floppylaufwerk SFD 1001 reagiert beim Befehl »I« (Initialisierung der Diskette) mit einer Fehlermeldung: READ ERROR auf Drive 1. Dies liegt daran, daß für die SFD 1001 das DOS 2.7 der Doppelfloppy 8250 nahezu unverändert übernommen wurde. Die 8250 initialisiert bei dem Befehl »I« beide Laufwerke (Drive 0 und 1). Will man ein einzelnes Laufwerk initialisieren, so ist der entsprechende Befehl »I0« oder »I1«. Nur bei dem korrekten Befehl »I0« gibt die SFD 1001 keine Fehlermeldung aus. Viele Programme verwenden aber den Sparbefehl »I« anstatt »I0« und fragen daraufhin den Fehlerkanal ab. Aus diesem Grund sendet das Betriebssystem bei dem Befehl »I« auch die dazugehörige »0« hinterher. Ein »Abstürzen« eines Programms aufgrund dieses Fehlers wird so vermieden. Wollen Sie dennoch beide Laufwerke gleichzeitig initialisieren, dann verwenden Sie dazu bitte den Befehl »I:«.

Am meisten wird wohl die Umschaltung der Zugriffsart seriell/parallel für die Gerätenummer 8 benötigt. Hier existiert neben dem Einstellen des Schalters 4 oder dem Setzen und Löschen des zweiten Bits eine dritte Möglichkeit. Durch gleichzeitiges Drücken der Tasten <CTRL + RETURN> wird das Bit 4 auf Null gesetzt und Bit 2 umgekehrt. Dies hat folgende Wirkung: Ein gelöschtes Bit 4 erlaubt die Abfrage der internen Bits anstelle der Schalter. Das Bit 2 (zuständig für die Gerätenummer 8) wird umgekehrt und damit der Zugriff von seriell auf parallel (oder umgekehrt) umgeschaltet. Diese Abfrage ist in der INPUT-Routine eingebunden. Sie können also während eines INPUT-Befehls die Übertragungsart für die Gerätenummer 8 ändern. Eine nützliche Hilfe.

Die Geräteadressen 10 und 11 bewirken übrigens stets eine serielle Übertragung, während 12 bis 15 eine parallele Übertragung bewirken.

Betriebssystem verbessert

Die Treibersoftware für den IEEE-488-Bus ist sowohl im C 64- wie im C 128-Betriebssystem eingebunden. Zu diesem Zweck mußten die Kassettenroutinen entfernt werden. Die RS232-Treiberroutinen unter der Gerätenummer 2 bleiben erhalten.

Sie sollten allerdings nicht mit Übertragungsraten über 1200 Bit/s arbeiten, wenn Sie gleichzeitig den parallelen Bus benutzen.



Stückliste

Integrierte Schaltkreise	Widerstände
1 74LS00; IC 1	1 Array 8 x 10 K; RN 1
1 µPD 8255 AC; IC 2	
1 SN75160; IC 3	**Sonstiges**
1 SN75161; IC 4	1 8fach DIP-Schalter; S1—S8
	1 IC-Fassung 40polig
Kondensatoren	2 IC-Fassung 20polig
1 x 15 µF, Tantal; C1	1 IC-Fassung 14polig
2 x 100 nF; C2, C3	

Tabelle 1. Die notwendigen Teile für das IEEE-488-Modul

Zuordnung der I/O-Leitungen

Port A	IEEE-488-Datenleitung	Port C	Steuerleitung/Schalter
PA0	DIO1	PC0	SW 8
PA1	DIO2	PC1	SRQ
PA2	DIO3	PC2	DAV
PA3	DIO4	PC3	EOI
PA4	DIO5	PC4	SW 7
PA5	DIO6	PC5	SW 6
PA6	DIO7	PC6	NRFD
PA7	DIO8	PC7	NDAC

Port B	Steuerleitung/Schalter
PB0	ATN
PB1	DC
PB2	TE
PB3	SW 1
PB4	SW 2
PB5	SW 3
PB6	SW 4
PB7	SW 5

Tabelle 2. Die Schalter und die Steuerleitungen an den Ports des µPD 8255

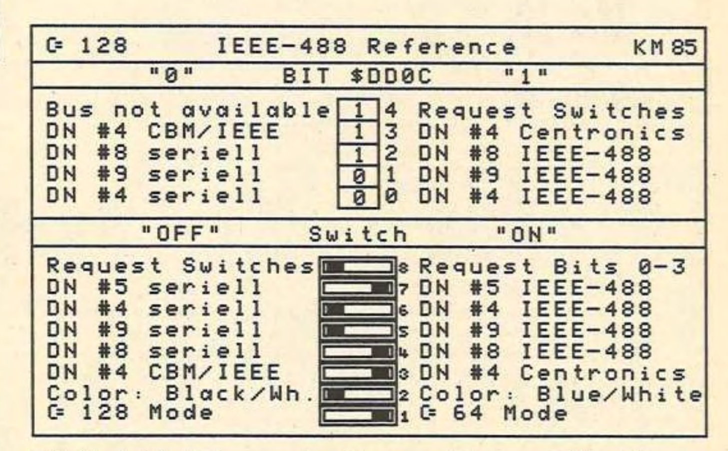
Im einzelnen sind im C 64-Modus folgende Verbesserungen implementiert:

1. <CTRL+*> bewirkt ein »OLD« für ein Basic-Programm nach einem NEW oder Reset.
2. <CTRL+DEL> löscht von der aktuellen Cursor-Position den Rest der Zeile.
3. Die linke <SHIFT>-Taste oder die Taste <SHIFT LOCK> bewirken eine Pausenfunktion für das Bildschirm-Scrolling. Die Ausgabe wird gestoppt, wenn eine neue Zeile am unteren Rand aufgebaut werden soll.
4. Die im Betriebssystem voreingestellte Gerätenummer für »LOAD« und »SAVE« ist von 1,1 (Kassette) auf 8,1 (Floppy) umgestellt worden. Davon unabhängig wird das Inhaltsverzeichnis mit 8,0 geladen.
5. Die Programmkennzeichnung »PRG« in einem Directory-Listing auf dem Bildschirm muß bei einem LOAD-Befehl nicht mehr beachtet werden. Es genügt, vor den Programmnamen ein »LOAD« zu schreiben und <RETURN> zu drücken.
6. Die Default-Nummer für den OPEN-Befehl ist auf x,8,15 gesetzt. Zum Öffnen des Befehlkanals der Floppystation genügt jetzt ein »OPEN1«.
7. Die Repeat-Funktion für die Tastatur ist eingeschaltet.
8. Die Tastenkombination <CTRL+←> schaltet den Quote-Mode (Gänsefüßchenmodus) ab. So können Sie dem Quote-Mode entkommen, ohne die Cursor-Position zu ändern.

Steckerbelegung CBM-IEEE und Standard-IEEE

Kontaktbenennung CBM	Standard-IEEE	Bezeichnung des Pins
1	1	DIO1
2	2	DIO2
Schlitz		
3	3	DIO3
4	4	DIO4
5	5	EOI
6	6	DAV
7	7	NRFD
8	8	NDAC
9	9	IFC
Schlitz		
10	10	SRQ
11	11	ATN
12	12	GND
A	13	DIO5
B	14	DIO6
Schlitz		
C	15	DIO7
D	16	DIO8
E	17	REN
F	18	GND (DAV)
H	19	GND (NRFD)
J	20	GND (NDAC)
K	21	GND (IFC)
Schlitz		
L	22	GND (SRQ)
M	23	GND (ATN)
N	24	GND (DIO1-8)

Tabelle 3. Vergleich der Steckerbelegung zwischen Commodore-IEEE und Standard-IEEE



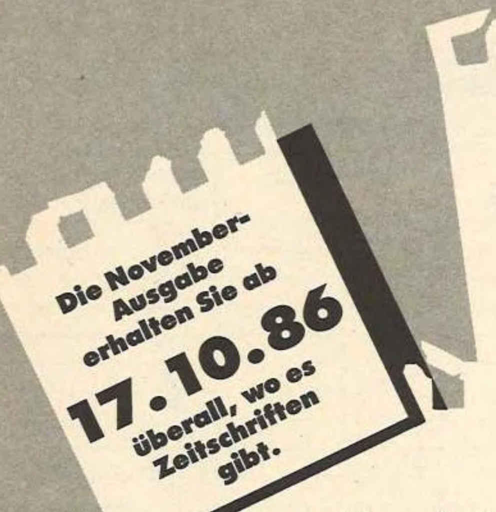
```
C= 128        IEEE-488 Reference          KM 85
        "0"       BIT $DD0C      "1"
Bus not available [1] 4 Request Switches
DN #4 CBM/IEEE    [1] 3 DN #4 Centronics
DN #8 seriell     [1] 2 DN #8 IEEE-488
DN #9 seriell     [0] 1 DN #9 IEEE-488
DN #4 seriell     [0] 0 DN #4 IEEE-488
        "OFF"      Switch        "ON"
Request Switches      8 Request Bits 0-3
DN #5 seriell         7 DN #5 IEEE-488
DN #4 seriell         6 DN #4 IEEE-488
DN #9 seriell         5 DN #9 IEEE-488
DN #8 seriell         4 DN #8 IEEE-488
DN #4 CBM/IEEE        3 DN #4 Centronics
Color: Black/Wh.      2 Color: Blue/White
C= 128 Mode           1 C= 64 Mode
```

Tabelle 4. Die Belegung des Steuerregisters und der DIP-Schalter

Tastencodes im C 64-Modus

ESC	liefert	CHR$(27)
TAB	liefert	CHR$(09)
ALT	liefert	CHR$(14)
HELP	liefert	CHR$(08)
LINE-FEED	liefert	CHR$(10)
NO SCROLL	liefert	CHR$(03)

Tabelle 5. Die Belegung der C 128-Tasten im C 64-Modus



Grund genug fürs neue 64'er

• Kompletter Schaltplan der Floppy 1541
in der 64'er-Ausgabe 11

• 2000 Mark zu gewinnen!
In unserem Wettbewerb: »Die schönste Grafik«

In der November-Ausgabe stellen wir vor,

Grafiken wie aus dem Großrechner

Ein Blick hinter die Kulissen der professionellen Filmgrafik bei Lucas-Film (»Star Trek«).

Wir zeigen, wie sich die Erkenntnisse aus Großrechner-Anlagen zur optimalen Programmierung von Grafik auf dem C 64 umsetzen lassen.

Eine große Übersicht mit Vergleich der besten Zeichen- und Malprogramme auf dem C 64 hilft beim Gelingen.

Maus kontra Joystick

Wie funktionieren sie?
Was sollte man beim Kauf beachten?
Welches Gerät eignet sich optimal zum Zeichnen?

... außerdem lesen Sie:

■ Mit dem Listing des Monats »3D-Grafik-Master« sind dreidimensionale Objekte auf dem C 64 kein Problem. ■ Ausführlicher Bericht von unserem Besuch der PCW-Messe in London mit den neuesten Trends und aktuellsten Produkten. ■ Im Einsteiger-Teil ausführliche Grundlagen für Grafikprogrammierung/Lexikon der Fachbegriffe/Literatur, die Anfänger unbedingt brauchen/Tips & Tricks von Profis für Anfänger. ■ Hardware-Bastelei: stereoklang mit 2-Sound Chips. So verdoppeln Sie die stimmgewalt des C 64. ■ Drucker im Test: Epson-LX86. ■ Anwendung des Monats: Deutsche Rechtschreibhilfe für Vizawrite. ■ Tips zu Giga-CAD, Geos und Master-Text.

■ Den großen Mailbox-Vergleichstest der Mailboxen in Deutschland. ■ Wir stellen die interessantesten Mailboxen Amerikas vor. ■ Sie erhalten eine ausführliche Marktübersicht von Akustikkopplern und Moden. ■ Akustikkoppler im Test: Was Sie leisten. ■ Datenfernübertragung: aber wie? ■ Das Listing des Monats: Der Soundmonitor. ■ Anwendung des Monats: Ein Sprachdigitizer zum Nachbauen. ■ Tips und Tricks zum Mastertext »Vizawrite« und »Glos« ■ Der große Grafikprogramm-Test.

Gutschein

FÜR EIN KOSTENLOSES PROBEEXEMPLAR DES 64'er-MAGAZINS

JA, ich möchte »64'er«, das Magazin für Computerfans, kennenlernen.
Senden Sie mir bitte die aktuellste Ausgabe kostenlos als Probeexemplar. Wenn mir »64'er« gefällt und ich es regelmäßig weiterbeziehen möchte, brauche ich nichts zu tun: Ich erhalte »64'er« dann regelmäßig frei Haus per Post und bezahle pro Jahr nur DM 78,— (Ausland auf Anfrage).

Vorname, Name

Straße | PLZ, Ort

Datum | 1. Unterschrift

Mir ist bekannt, daß ich diese Bestellung innerhalb von 8 Tagen bei der Bestelladresse widerrufen kann und bestätige dies durch meine zweite Unterschrift. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs.

Datum | 2. Unterschrift

Gutschein ausfüllen, ausschneiden, in ein Kuvert stecken und absenden an:
Markt&Technik Verlag Aktiengesellschaft, Vertrieb, Postfach 1304, 8013 Haar

64S1086

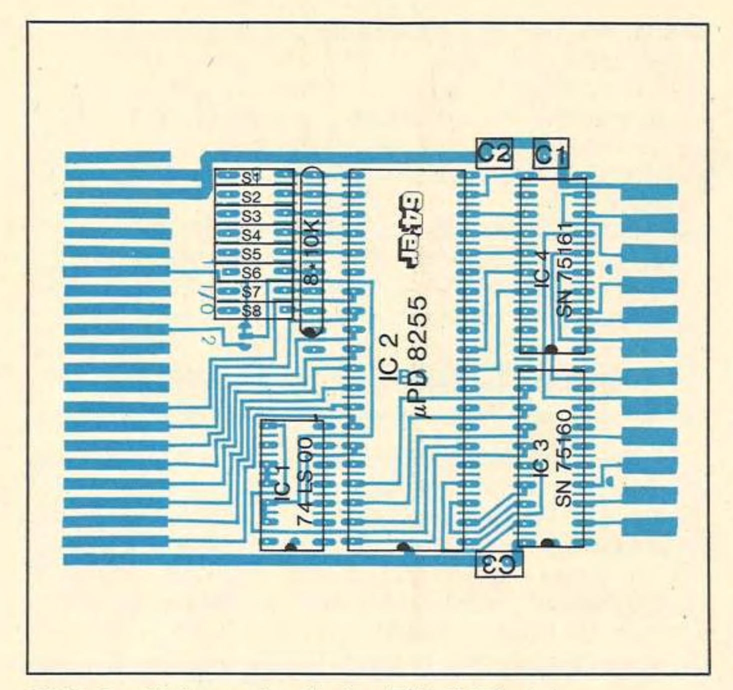
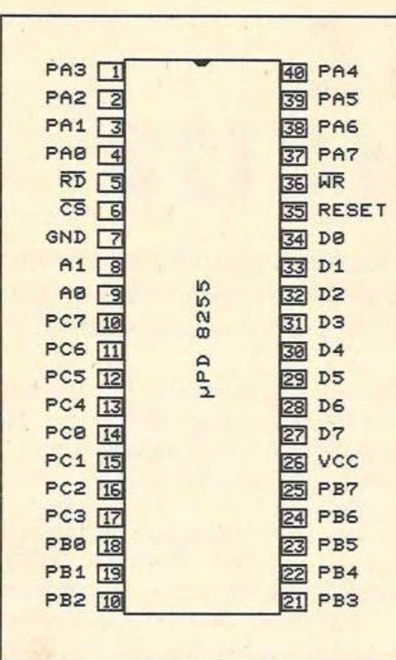


Bild 4. Bestückungsplan für das IEEE-488-Modul



Dieses IC ist das Herz der Schaltung. Es ist ein programmierbarer Interface-Baustein, der mit 2 MHz betrieben werden kann. Geschwindigkeitsprobleme wird es daher nicht geben.

Bild 5. Die Anschlüsse des Bausteins µPD 8255

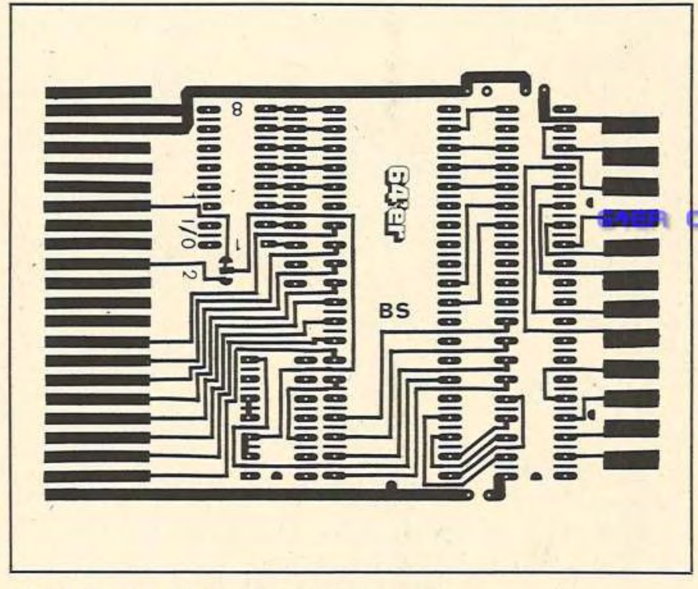


Bild 6. Das Layout der Bestückungsseite

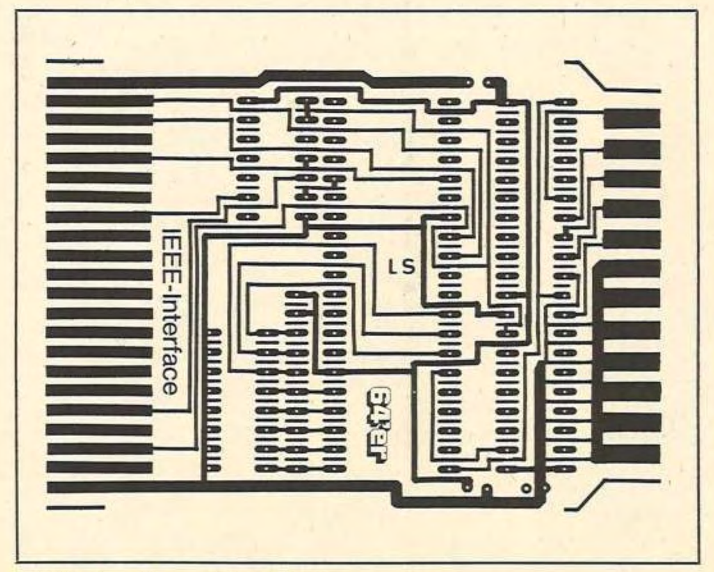


Bild 7. Das Layout der Lötseite



Bild 8. Der Commodore 128 D mit einem Doppellaufwerk 8250 LP

9. Die Tastenfolge <CBM+RUN/STOP> ergibt ein »RUN:«
10. <SHIFT+RUN/STOP> bewirkt das Laden und Starten des ersten Programms vom Diskettenlaufwerk (LOAD"0:*":RUN).
11. Die Tastenkombination <F1+F3+DEL> unterdrückt die Ausführung des Autostart-Codes »cbm80« beim Reset oder Einschalten. Reset-feste Programme können Sie so verlassen.
12. <F1+DEL> wandelt beim Reset den Autostart-Code »cbm80« im RAM bei $8004 bis $8008 in »ibm80« um. Im Gegensatz zur Methode <F1+F3+DEL> ist danach ein NMI-Warmstart des Computers (<RUN/STOP+RESTORE>) möglich.
13. Die neuen Cursor-Steuertasten der C 128-Tastatur funktionieren auch im C 64-Modus.
14. Der Zehnerblock des C 128 ist auch im C 64-Modus zugänglich. Die Tasten liefern den gleichen Code wie die entsprechenden Tasten der normalen C 64-Tastatur. Dies gilt auch im SHIFT-Modus.
15. Die Funktion der zusätzlichen Tasten des C 128 für den C 64-Modus können Sie der Tabelle 5 entnehmen. Die übrigen Zusatztasten bleiben ohne Funktion.

In Bild 8 sehen Sie den C 128 D gemeinsam mit einem Doppellaufwerk 8250 LP im Einsatz. Alle, die sich den Selbstbau nicht zutrauen, können das fertige IEEE-Interface auch bei Roßmöller, Maxstraße 50 - 52, 5300 Bonn 1, für 198 Mark erhalten.

(K. Mandelatz/kn)

Formel C für den C 128

Floppy-Speeder, Maschinensprache-Monitor, Grafik- und Toolkit-Befehle - dies alles bietet Ihnen Formel C in einem einzigen Steckmodul.

Viele Erweiterungen für den C 128 im C 64-Modus sind in ihren Leistungen mehr oder weniger beschränkt, da sie nur auf einen bestimmten Anwendungsbereich zugeschnitten sind. Hier bietet »Formel C« (Bild 1) eine reizvolle Alternative.

Beim Arbeiten im C 64-Modus ist es immer von Vorteil, verschiedene Toolkit-Funktionen wie zum Beispiel AUTO, RENUMBER, FIND oder OLD zur Verfügung zu haben. Formel C ist aber wesentlich mehr als nur eine Befehlserweiterung für den C 64-Modus. Denn hier wurden außer Toolkit-Funktionen (siehe Tabelle) zur Unterstützung der Programmierung noch ein Maschinensprache-Monitor, ein Disketten-Monitor, ein komfortabler 2-Pass-Assembler und einige schnelle Grafik-Befehle eingebaut. Erwähnenswert ist der im Modul integrierte Floppy-Beschleuniger, der allerdings nur mit dem Diskettenlaufwerk 1541 zusammenarbeitet. Möchte man dieses Modul in einen C 128D einbauen, ist das interne Laufwerk 1571 abzuschalten und eine 1541 extern anzuschließen. Dafür sind dann aber auch die Geschwindigkeiten bei LOAD, SAVE und VERIFY etwa 16mal schneller als bei einer nicht umgerüsteten 1541. Und das in allen drei Modi (C 64, C 128 und CP/M). Ein Backup- und Filecopy-Programm sowie eine gegen Aufpreis erhältliche Centronics-Schnittstelle runden das Bild von Formel C angenehm ab.

Die für diese Funktionen benötigte Software ist in einem 16-KByte- und einem 32-KByte-EPROM untergebracht, zwischen denen ständig hin- und hergeschaltet wird.

Folgende Funktionen können durch Druck auf bestimmte Tasten beim Einschalten erzielt werden:

Keine Taste: Der Computer verhält sich so, als ob kein Modul eingesteckt wäre.

<CTRL>: Der Computer springt in den C 64-Modus. Mit dem Befehl OFF kann nun der normale C 64-Modus ohne Formel C-Unterstützung gewählt werden.

<←>: Eine eingelegte CP/M-Diskette bootet mit erhöhter Geschwindigkeit (5fach). Nach Abschluß des Bootens führt das Diskettenlaufwerk CP/M-Operationen mit 15facher Geschwindigkeit aus.

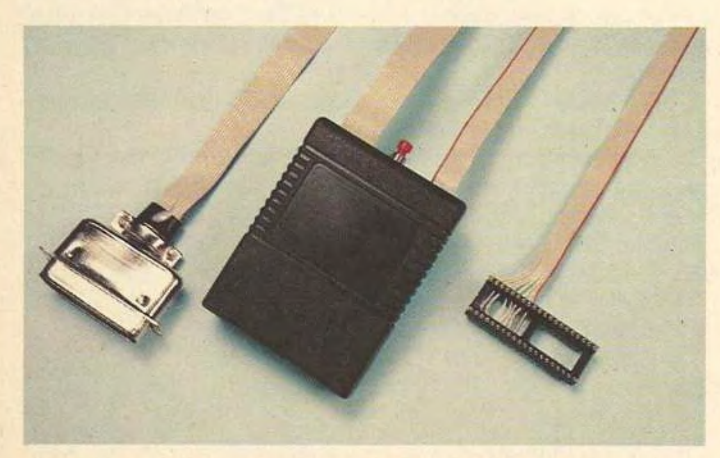


Bild 1. Formel C - eine gelungene Erweiterung für den C 64-Modus des C 128

Besonderheiten im CP/M-Modus:
- Der Befehl BOOT führt in jedem Fall zum Laden des CP/M-Systems mit normaler Geschwindigkeit. Das beschleunigte Arbeiten ist nur durch einen Reset mit gedrückter <CTRL>-Taste zu erreichen.
- Im CP/M-Modus steht eine schnelle Formatier-Routine zur Verfügung.
- Komplette CP/M-Disketten können im C 64-Modus mit dem Befehl BACKUP in kurzer Zeit kopiert werden.
- Mit einem zweiten Floppy-Kabel lassen sich zwei Laufwerke unter CP/M betreiben.

Floppy-Beschleuniger eingebaut

Der Floppy-Beschleuniger ist an einem Kabel erkennbar, das aus dem Modul herausgeführt ist. Der Adaptersockel ist in das Laufwerk einzubauen. Dort muß lediglich ein Chip auf der Platine herausgezogen und der Sockel eingesetzt werden. Weitere Umbauarbeiten gibt es nicht. Die Lade- und Speichergeschwindigkeit steigt damit um den Faktor 6 bis 8. Verwendet man aber speziell formatierte Disketten, läßt sich diese Geschwindigkeit auf das 13- bis 16fache steigern.

Hier nun die Leistungen in einer tabellarischen Übersicht:

Floppyfunktionen	
LOAD	
SAVE	in allen drei Modi
VERIFY	16mal schneller
Formatieren	etwa 18 Sekunden
Backup	etwa 80 Sekunden
D[illegible] Befehle (C 64-Modus)	
DLOAD	
DVERIFY	
DSAVE	
DAPPEND	Diskettenbezogene Befehle
DIR	Anzeige des Disketten-Directories ohne Programmverlust
LRUN	Laden und Starten eines Programms
REPLACE	Korrigiertes Überschreiben eines Programms auf Diskette
STATUS	Anzeige des Laufwerks-Status
HEADER	Formatieren
BACKUP	Kopieren einer ganzen Diskette
SEND	Befehle an Laufwerk senden
FILECOPY	Kopieren einzelner Dateien
Toolkit-Funktionen (C 64-Modus)	
AUTO	Automatische Zeilennummern-Vorgabe
RENUMBER	Umnumerierung eines Basic-Programms
DELETE	Löschen von Programm-Teilbereichen
DEZ	Umrechnung
HEX	verschiedener Zahlensysteme
FIND	Findet Zeichenketten oder Befehle
JUMP	Führt Maschinenprogramme aus
OLD	Retten eines gelöschten Programms
RESET	Reset mit Modul-Neustart
LPRINT	Ausgabe
LLIST	auf Drucker
HARDCOPY	über volle DIN-A4-Breite
HELP	Übersicht über die neuen Befehle
OFF	Reset ohne Modul-Neustart
TKITOFF	Abschalten der Befehlserweiterung
Maschinensprache-Monitor	
Ilegale Opcodes	
Memory-Dump	Hex oder ASCII
Disassemble	Disassemblieren

Tabelle. Formel 64 im Überblick

Diese speziell formatierten Disketten können aber auch ohne Modul gelesen werden. Auch der Formatierungsvorgang selbst hat sich bezüglich der Geschwindigkeit verbessert. Gegenüber 70 Sekunden beim Original-Betriebssystem ist eine Diskette nun in etwa 18 Sekunden formatiert. Kritische Programme wie Textomat plus oder Summer Games II liefen einwandfrei.

Formel C hat für Basic- oder Maschinensprache-Programmierer im C 64-Modus einiges zu bieten. Es existieren Befehle wie RENUMBER zum Umnumerieren von Basic-Programmen genauso wie FIND, um einzelne Zeichenfolgen zu suchen. HCOPY erzeugt eine Hardcopy des Bildschirms, egal, ob vom Text- oder Grafikbild. Allerdings nur auf einem Epson-kompatiblen Drucker. Die Hardcopy wird über das gesamte Querformat einer DIN-A4-Seite gedruckt. Des weiteren gibt es Kommandos wie LLIST oder LPRINT zum einfachen Ansteuern eines Druckers. Der Befehl DELETE löscht ganze Bereiche eines Programms. Abgerundet werden diese Toolkit-Funktionen durch Befehle wie OLD, HELP, HEX und DEZ.

Für Assemblerfreaks ist ein komfortabler Monitor mit integriertem Diskmonitor eingebaut. Vorhanden sind Befehle wie MEMORY, ASSEMBLE, DISASSEMBLE, FIND und VERIFY.

Assemble	Assemblieren
Register	Anzeige der Prozessor-Register
Single-Step	Schrittweises Abarbeiten von Assembler-Programmen
Goto	Startet ein Maschinenprogramm
Exit	Führt zurück ins Basic
Printer on	Leitet Ausgaben auf Drucker
Printer off	Schaltet zurück auf Bildschirm
Find	Findet Befehle
Fill	Füllt Speicherbereich mit bestimmtem Wert
Hunt	Findet Bitkombinationen
Breakpoint	Setzt Abbruchbedingung bei Step
Quickstep	Ähnlich Single-Step
Transfer	Verschiebt Speicherbereiche
Compare	Vergleicht Speicherbereiche
Load	Lädt ein Maschinenprogramm
Save	Speichert ein Maschinenprogramm
Disk-Monitor	Mit Read-Write-Sektor

2-Pass-Assembler
Befehlskompatibel zu Profi-Ass
Zusätzliche Label-Tabelle beliebig verschiebbar
Verarbeitung illegaler Opcodes

Grafik	
HIRES	Schaltet hochauflösende Grafik ein
MULTI	Schaltet Mehrfarben-Grafik ein
FRAME	Setzt die Rahmenfarbe
CLEAR	Löscht den Grafikbildschirm
TEXT	Umschalten auf Text-Modus
PAGE	Umschalten verschiedener Grafikseiten
COPY	Kopiert Bildschirme
MIX	Mischt Grafikbilder
INVERT	Invertieren der Grafik
GSAVE	Speichert Grafikbilder
GLOAD	Lädt Grafikbilder
PLOT	Setzt einen Grafikpunkt
LINE	Zieht eine Linie
DRAW	Zeichnen unregelmäßiger Linien
BOX	Zeichnet ein Rechteck
BLOCK	Zeichnet ein ausgefülltes Rechteck
CIRCLE	Erstellt einen Kreis
FILL	Füllt eine Fläche aus
HPRINT	Schreiben von Text
VPRINT	in den Grafikbildschirm

Interessant ist der vorhandene Einzelschrittmodus, der es erlaubt, den Ablauf von Maschinenroutinen genau zu beobachten. Außergewöhnlich ist die Unterstützung von illegalen Opcodes des 8502-Prozessors.

Der Diskmonitor erlaubt umfangreiche Operationen im Laufwerk-RAM und Manipulationen einzelner Blocks auf der Diskette. Der vorhandene Assembler ist befehlskompatibel zum Profi-Ass. Es wurden allerdings einige zusätzliche Befehle mit aufgenommen. So verarbeitet er ebenfalls illegale Opcodes.

Grafik

Die eingebauten Grafikbefehle für den C 64-Modus überzeugen durch ihre Schnelligkeit und Leistungsfähigkeit. Es werden bis zu vier Grafikseiten verwaltet, bei denen die Möglichkeit besteht, diese untereinander frei zu kopieren und zu mischen. Die erste Grafikseite liegt im RAM unter dem Basic-Interpreter. Alle weiteren Grafikseiten verbrauchen, insofern sie benötigt werden, Basic-Speicherplatz. Grafikfunktionen sind reichlich vorhanden. So existieren Befehle wie CIRCLE und BOX zum Zeichnen von Ellipsen, Kreisen oder Vierecken, LINE zum Ziehen von Linien und DRAW zum Entwerfen komplexer Gebilde. Text kann sowohl horizontal wie auch vertikal in eine Grafik eingebaut werden. PLOT setzt oder löscht einzelne Punkte. Mit GSAVE und GLOAD ist das Speichern und Laden einzelner Grafikseiten möglich, wobei dann beim Laden sofort auf hochauflösende Grafik geschaltet wird.

Centronics-Schnittstelle optional

Gegen einen Aufpreis von 49 Mark erhält man eine Centronics-Schnittstelle, mit der man einen beliebigen Centronics-Drucker anschließen kann. Über die Sekundäradresse können dabei verschiedene Modi wie Listingdruck, Linearkanal und Textdruck eingestellt werden. Formel C erkennt automatisch, an welchem Bus ein Drucker angeschlossen ist. Entsprechend wird dann dieser Drucker angesteuert. Die automatische Erkennung ist auch abschaltbar.

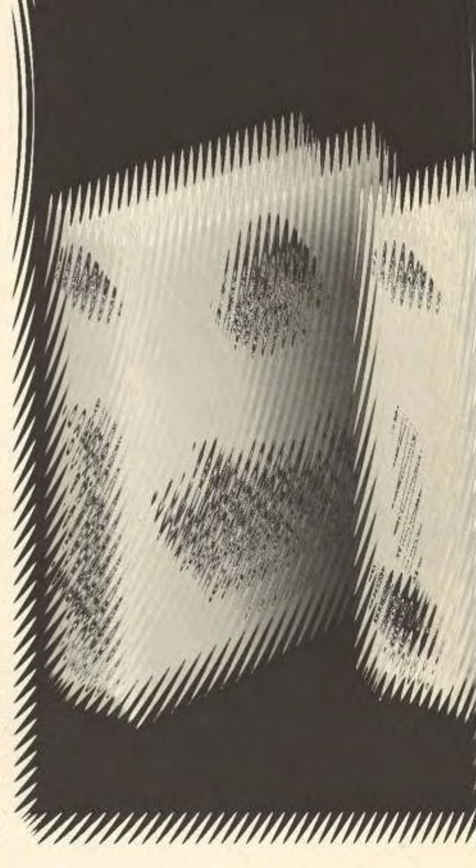
Natürlich wurden bei all diesen Erweiterungen einige Diskettenbefehle nicht vergessen. So läßt sich mit DIR ein Directory von einer Diskette lesen, der Disketten-Status abfragen und auch DOS-Befehle können gesendet werden. Außergewöhnlich sind die Funktionen, die über die <RESTORE>-Taste erreichbar sind. Es läßt sich damit jederzeit eine Hardcopy des Bildschirmes starten, häufig auch innerhalb laufender Programme oder Spiele. Weiterhin eingebaut ist eine BACKUP-Routine, die eine vollständige Kopie einer Diskette in 80 Sekunden erstellt. Auch einzelne Dateien und Programme lassen sich mit FILECOPY kopieren.

Fazit

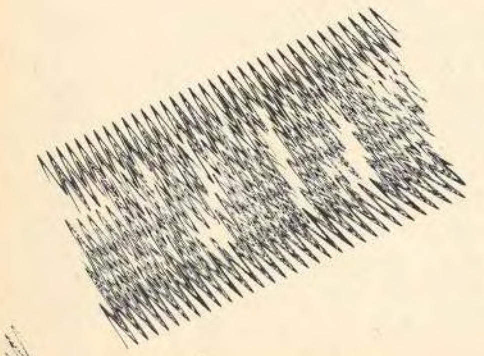
Formel C ist eine gelungene Erweiterung für den C 128. Speziell für den C 64-Modus sind sehr viele gute Erweiterungen in einem Gehäuse zusammengefaßt. Außerdem erhöht das Modul die Datenübertragungsgeschwindigkeit von und zum Diskettenlaufwerk. Leider kann es nur mit der Diskettenstation 1541 betrieben werden. Für den Preis von 198 Mark ist es aber auf jeden Fall eine Erweiterung, die sehr viele und umfassende Funktionen in einem Modul beinhaltet und jedermann empfohlen werden kann, der oft im C 64-Modus arbeitet und häufig benötigte Erweiterungen sucht. (dm)

Info: Grewe Computertechnik, Wiesenstr. 82, 4350 Recklinghausen, Tel. 02361/181354


Unschl

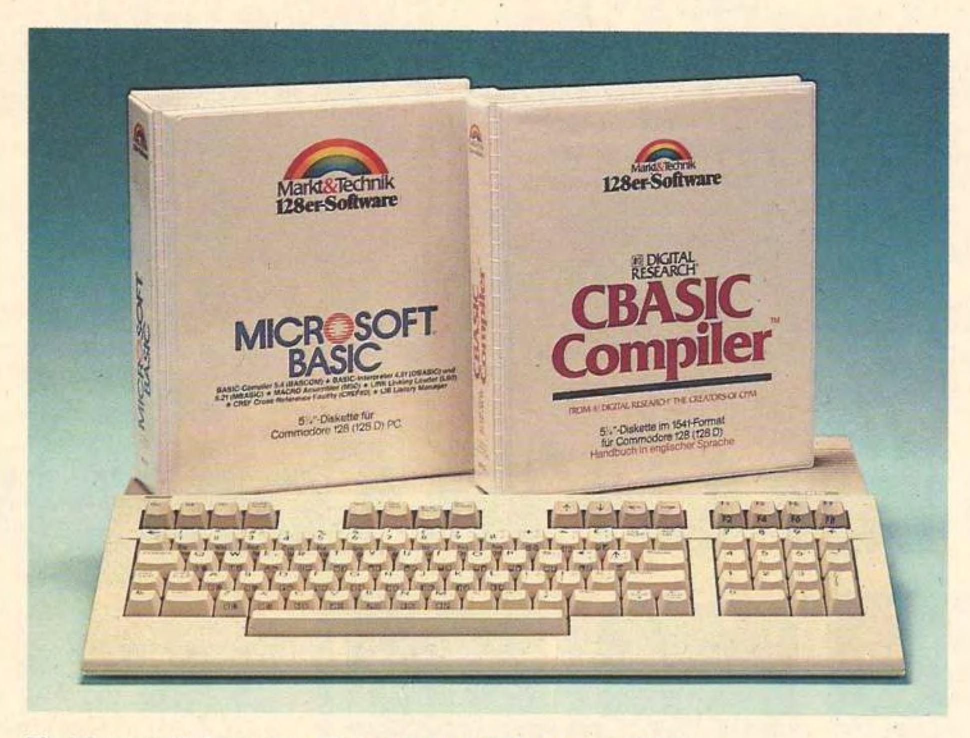







agbar!

Basic unter CP/M: MS- und CBasic auf dem Commodore 128 PC



Ein leistungsfähiges Duo: Microsoft- und CBasic unter CP/M

Sinnvolles Arbeiten unter CP/M beginnt entweder mit Programmen wie dBase II, Multiplan und Wordstar 3.0, oder wendet sich an den Anwender, der selbst programmieren möchte. Dies ist beispielsweise mit den beiden Basic-Compilern/Interpretern Microsoft-Basic und CBasic möglich.

Nachdem man den C 128 ausgepackt und eine Weile mit der CP/M-Systemdiskette herumexperimentiert hat, stellt man oft enttäuscht fest, was CP/M eigentlich ist: Ein Betriebssystem. Es stellt lediglich einen Grundstock dar, auf dem Programme wie etwa dBase II oder auch Compiler und Interpreter aufbauen. Da die Programmiersprache Basic sehr verbreitet ist, stellen wir Ihnen zwei Basic-Compiler/Interpreter vor und zeigen, was man mit ihnen anfangen kann.

MS-Basic – das Profisystem

Den MS-Basic-Interpreter und Compiler gibt es jetzt mit einigen Zusatzprogrammen vom Markt&Technik Verlag auf einer Diskette für den Commodore 128 PC. Die Zusatzprogramme, die den Wert des Interpreters fast übertreffen, bestehen aus einem Compiler, sowie einem Z80-Makroassembler der Spitzenklasse. Der Lieferumfang umfaßt ein etwa 700seitiges Begleitbuch sowie eine doppelseitig beschriebene Diskette. Auf der Diskette befinden sich drei unabhängige Programmsammlungen, der Basic-Interpreter in zwei Ausführungen, der Compiler mit Linker und Bibliothek sowie der Makroassembler mit Makrobibliothek und Crossreferenz-Programm.

Der Interpreter wird unter CP/M durch Eingabe von »MBASIC« gestartet. Er meldet sich dann mit der Angabe der frei verfügbaren Bytes. Hier erscheinen maximal etwa 35000 Bytes. Das erinnert stark an verflossen geglaubte C 64-Dimensionen. Ganz so schlimm, wie es scheint, ist es aber doch nicht, da vieles durch den sehr großen Sprachumfang des Interpreters aufgefangen wird. Hier spürt man, daß MS-Basic eine professionelle Sprache ist. Für die Dateiverwaltung finden sich sämtliche hierzu benötigten Befehle, aber auch Programmierprofis kommen auf ihre Kosten. Es wird zum Beispiel die Möglichkeit angeboten, mit Zahlen doppelter Genauigkeit zu rechnen. Befehle wie »IF..THEN..ELSE« oder »PRINT USING« gehören zum Standard. Insgesamt werden fast 140 Befehle und Funktionen angeboten (siehe hierzu auch die Befehls- und Funktionsübersicht in Tabelle 1).

Als gravierendster Unterschied zum Basic-Interpreter des Commodore 128, der ja ebenfalls von Microsoft stammt, fällt auf, daß zwischen zwei Befehlen ein Zwischenraum stehen muß. Dies ermöglicht die Verwendung von beliebig langen Variablennamen, wobei jeder Buchstabe unterscheidungsrelevant ist. Es lassen sich dadurch Variablennamen verwenden, die direkt deren Funktion im Programm beschreiben. Viele Programmierer werden die einfache Editiermöglichkeit beim Editor des C 128 vermissen, denn im MS-Basic gibt es nur den »EDIT«-Befehl, der etwas umständlich zu handhaben ist. Das bietet bei genauerer Betrachtung jedoch einen großen Vorteil. Dem Programmierer wird es dadurch schwer gemacht, grundlegende Änderungen nachträglich in einem Programm durchzuführen und so dem gefürchteten »Spaghetticode« zu verfallen. Wer gleich eine Programmplanung vornimmt, braucht solche Änderungen nicht durchzuführen, das ganze Programm wird dadurch sehr übersichtlich. Stellt man allerdings Geschwindigkeitsvergleiche mit dem eingebauten Basic-Interpreter des C 128 an, so wird man enttäuscht. Die Ausführungszeit von MS-Basic-Programmen ist drei- bis viermal langsamer, als die der Programme im

C 128-Modus. Dieser Umstand ist aber nicht dem Interpreter anzulasten, sondern dem CP/M-System des Commodore. Hier tritt wieder das alte Problem in Erscheinung, daß der Z80 im Commodore 128 sehr niedrig getaktet wird.

Abhilfe versprechen sowohl der umfangreiche Befehlssatz, der viele Unterprogramme erspart, sowie der mitgelieferte Compiler. Zusammen mit dem Linker verwandelt er Basic-Programme in direkt ausführbare Programme, die unter CP/M wie gewohnt durch Angabe des Namens aufgerufen und gestartet werden. Hierbei ist gegenüber den nicht compilierten Programmen ein maximaler Geschwindigkeitszuwachs von 30% zu verzeichnen. Bei ausgabeintensiven Programmen ist allerdings kaum eine Verbesserung zu erreichen, da die Geschwindigkeit hier stark vom Computer abhängt.

Die zweite Version des Interpreters ist eine früher datierte Ausgabe von MS-Basic. Die Schreibweise der Befehle gleicht der von Basic 7.0; Variablennamen sollen nur zwei Zeichen umfassen und es gelten die üblichen Einschränkungen, daß in ihnen zum Beispiel keine Schlüsselworte enthalten sein dürfen. Ebenso hat diese Version einen eingeschränkten Befehlsumfang. Mitgeliefert wird sie hauptsächlich, um Programme, die auf anderen CP/M-Computern mit diesem Interpreter geschrieben wurden, leichter übertragen zu können.

Gibt es Probleme, die sehr zeitintensiv sind, oder sollen Betriebssystemveränderungen gemacht werden, so wird man vom Microsoft-Programmpaket nicht im Stich gelassen. Hierfür bietet sich der ausgezeichnete Assembler an, der mit zum Besten gehört, was für Z80-Prozessoren derzeitig verfügbar ist. Dieser Assembler entschädigt für den unverständlichen Zug von Commodore, die im Handbuch erwähnten und beschriebenen Assemblerdienstprogramme MAC, LINK, LIB etc., auf einer extra für 80 Mark zu erwerbenden Diskette zu vertreiben. Hier sei gleich davor gewarnt, mit dem Microsoft-M80-Assembler Maschinensprache-Programmierung lernen zu wollen. Für diese Zwecke ist das ganze Paket viel zu umfangreich, und der Anfänger würde von den vielen Möglichkeiten nur verwirrt. Absolute Neulinge sollten auf einem einfacheren Assembler die ersten Schritte wagen. Der Profi aber bekommt ein Werkzeug in die Hand, das er nach kurzer Zeit nicht mehr missen möchte. Von Makro-Definition über Bibliotheken bis hin zur bedingten Assemblierung ist alles vorhanden, was zum Arbeiten mit Z80-Prozessoren notwendig ist. Leider stellt Microsoft keinen Editor zur Verfügung, so daß auf den mit der Commodore-Systemdiskette mitgelieferten Zeileneditor »ED« zurückgegriffen werden muß. Für größere Projekte ist aber ein großer Editor wie Wordstar aufgrund der einfacheren Bearbeitung von Programmen sehr zu empfehlen.

Ein wichtiges Werkzeug zur Fehlersuche wird mit dem Programm CREF mitgeliefert. Es bietet sich so die Möglichkeit, eine alphabetisch geordnete Liste aller im Assemblerlisting vorkommenden Variablen mit Zugriffsangabe anzulegen, ebenso sind Sprünge im Programm einfach nachzuvollziehen.

Reservierte Wörter in MS-Basic

ABS	ERASE	LOG	RIGTH$
AND	ERL	LPOS	RND
ASC	ERR	LPRINT	RSET
ATN	ERROR	LSET	RUN
AUTO	EXP	MERGE	SAVE
CALL	FIELD	MID$	SGN
CDBL	FILES	MKD$	SIN
CHAIN	FIX	MKI$	SPACE$
CHR$	FN	MKS$	SPC
CINT	FOR	MOD	SQR
CLEAR	FRE	NAME	STEP
CLOSE	GET	NEW	STOP
COMMON	GOSUB	NEXT	STR$
CONT	HEX$	NOT	STRING$
COS	IF	DCT$	SWAP
CSAVE	IMP	ON	SYSTEM
CSNG	INKEY$	OPEN	TAB
CVD	INP	OPTION	TAN
CVI	INPUT	OR	THEN
CVS	INPUT#	OUT	TO
DATA	INPUT$	PEEK	TROFF
DEFDBL	INSTR	POKE	TRON
DEFINT	INT	POS	USING
DEFSNG	KILL	PRINT	USR
DEFSTR	LEFT$	PUT	VAL
DEF USR	LEN	RANDOMIZE	VARPTR
DELETE	LET	READ	WAIT
DIM	LINE	REM	WEND
EDIT	LIST	RENUM	WHILE
ELSE	LLIST	RESET	WIDTH
END	LOAD	RESTORE	WRITE
EOF	LOC	RESUME	WRITE#
EQV	LOF	RETURN	XOR

Tabelle 1. Sprachumfang MS-Basic

Der Einsatz von MS-Basic

Insgesamt betrachtet stellt das Microsoft-Basic Paket eine abgerundete Programmsammlung dar, die vielen Anwendern Nutzen bringt. Der Basic-Interpreter ist in Zusammenhang mit dem Compiler ideal, um Programme zu schreiben, die mit fertigen CP/M-Programmen zusammenarbeiten. So ist es zum Beispiel möglich, eine einfache Adreßverwaltung zu schreiben, die die Daten im Wordstar-Mailmerge-Format auf Diskette ablegt oder um Multiplan-Datenbestände grafisch auf Druckern auszuwerten. Sich das Programmpaket zu kaufen, um nur alternativ unter CP/M Basic zu programmieren, ist nicht sehr sinnvoll. Der eingebaute Interpreter ist billiger und in den meisten Fällen auch besser; ebenso tun sich Anfänger leichter, als mit MS-Basic, da beim Basic 7.0, wie oben erwähnt, Verbesserungen ganz einfach anzubringen sind.

Das Microsoft-Paket ist also mehr als Bindeglied zwischen vorhandenen CP/M-Programmen und zur Lösung individueller Problemstellungen unter CP/M anzusehen. In dieser Aufgabe muß es sich allerdings Vergleiche mit anderen Sprachen, wie zum Beispiel Turbo-Pascal, gefallen lassen. MS-Basic hat den großen Vorzug, Professionalität, Einfachheit und großen Bekanntheitsgrad der Sprache miteinander zu verbinden. Die Sprache Basic ist zwar eine einfach zu erlernende Computersprache, das Paket bietet aber auch wachsenden Ansprüchen etwas, wobei hier insbesondere der Assembler gemeint ist. Durch dessen hervorragende Qualität ist die Programmsammlung auch als Maschinensprache-Entwicklungspaket zu sehen. Die Hauptteile können mit dem Assembler geschrieben werden, Test- und Überwachungsprogramme in MS-Basic, wobei durch den Compiler kaum Geschwindigkeitseinbußen hingenommen werden müssen.

Leider stellt das mitgelieferte Handbuch hohe Ansprüche an die Englischkenntnisse des Benutzers. Nur die ersten 160 Seiten des etwa 700 Seiten starken Druckwerks sind in Deutsch und bieten eine kurze Übersicht über die Möglichkeiten von MS-Basic. Möchte man ausführliche Informationen über Befehle oder Besonderheiten, so muß man dies im englischen Teil nachlesen, der dafür sehr ausführlich gehalten ist.

Fazit: Das Haupteinsatzgebiet des MS-Basic-Paketes von Microsoft liegt beim erfahrenen CP/M-Anwender. Dieser Benutzerkreis bekommt ein Programmentwicklungspaket zur Verfügung gestellt, das auch professionelle Wünsche befriedigt. Mit diesem Werkzeug besitzt der Programmierer

Reservierte Wörter in CBasic

ABS	ERR	LE	PUBLIC	STRING$
AND	ERRL	LEFT$	PUT	SUB
AS	ERROR	LEN	RANDOMIZE	TAB
ASC	ERRX	LET	READ	TAN
ATTACH	EQ	LINE	READONLY	THEN
ATN	EXP	LOCK	REAL	TO
BUFF	EXTERNAL	LOCKED	RECL	UCASE$
CALL	FEND	LOG	RECS	UNLOCK
CHAIN	FLOAT	LPRINTER	REM	UNLOCKED
CHR$	FOR	LT	REMARK	USING
CLOSE	FRE	MATCH	RENAME	VAL
COMMAND$	GE	MFRE	RESTORE	VARPTR
COMMON	GET	MID$	RETURN	WEND
CONCHAR%	GO	MOD	RIGHT$	WHILE
CONSOLE	GOSUB	NE	RND	WIDTH
CONSTAT%	GOTO	NEXT	SADD	XOR
COS	GT	NOT	SGN	%CHAIN
CREATE	IF	ON	SHIFT	%DEBUG
DATA	INITIALIZE	OPEN	SIN	%EJECT
DEF	INKEY	OR	SIZE	%INCLUDE
DELETE	INP	OUT	SQR	%LIST
DETACH	INPUT	PEEK	STEP	%NOLIST
DIM	INT	POKE	STOP	%PAGE
ELSE	INT%	POS	STR$	
END	INTEGER	PRINT	STRING	

Tabelle 2. Sprachumfang CBasic

alles, was er zur optimalen Nutzung des Betriebssystems CP/M benötigt. Unerfahrenen Computerbesitzern kann man MS-Basic nicht uneingeschränkt empfehlen, es besteht leicht die Gefahr, von den vielfältigen Möglichkeiten »überrollt« zu werden. Aber gerade durch seine Vielfältigkeit schließt MS-Basic eine wichtige Lücke in der Programmsammlung für den C128.

CBasic – Komfort ist alles

Mit CBasic bietet Markt&Technik einen weiteren Basic-Compiler an, der vor allem durch die komfortable Programmierumgebung hervorsticht. Auf der Diskette befinden sich der Compiler, ein Linker und ein Programm zur Verwaltung von Modulen. Was es damit auf sich hat, erfahren Sie später.

Um nun ein CBasic-Programm zu schreiben, muß erst einmal der Quelltext erfaßt werden. Hier muß auf den CP/M-Editor zurückgegriffen werden, dessen Leistung allerdings nicht mehr dem heutigen Standard entspricht. Wer über anwendungsfreundlichere Editoren verfügt, zum Beispiel Wordstar, sollte diese auf jeden Fall benutzen.

Zum Basic-Interpreter des C128-Modus weist CBasic einige Unterschiede auf, die sofort positiv auffallen. So braucht der Programmierer keine Zeilennummern mehr zu verwenden, sondern kann direkt Labels, also konkrete Namen, für die Sprungziele angeben. Natürlich können auch die herkömmlichen Zeilennummern verwendet werden. Des weiteren ist es möglich, in CBasic Funktionen zu definieren. Aber das kennen wir ja schon vom C128-Modus, werden Sie jetzt vielleicht sagen. Die CBasic-Funktionen, und das ist der große Unterschied, können mehrzeilig sein und wie Unterprogramme behandelt werden. Von herkömmlichen Unterprogrammen (GOSUB) unterscheiden sie sich durch die Möglichkeit, Variablen an die Funktion zu übergeben, ähnlich den Funktionen in der Sprache C. Die Variablen, die dann in der Funktion benutzt werden, sind lokal, das heißt, sie gelten nur für den Bereich der Funktion.

Bei der Variablenbehandlung wäre noch eine Besonderheit von CBasic erwähnenswert. Zwar definiert der Compiler jede Variable bei ihrem ersten Aufruf im Programm selbst, doch besteht auch die Möglichkeit, den Variablentyp am Programmanfang festzulegen. Es können Integer-, String- und Real-Variablen deklariert werden. Der Vorteil dieses Verfahrens liegt im Weglassen der näheren Variablenbezeichnung (%,$) im weiteren Programmverlauf.

Doch damit ist der Sprachschatz von CBasic (Tabelle 2) noch nicht erschöpft. Mit einem speziellen Befehl können auch Overlays, also Programme, die sich gegenseitig nachladen, erzeugt werden. Dem nachgeladenen Programm übergibt CBasic alle Variable, die mit der Anweisung »COMMON« deklariert wurden. Auch diese Variable können vorher mit den bereits bekannten Anweisungen auf ihren Typ voreingestellt sein. Um die ganze Sache noch abzurunden, existieren selbstverständlich noch Anweisungen wie »WHILE« und »IF ... THEN ... ELSE«. Des weiteren verfügt CBasic noch über einige Funktionen zum Ermitteln von Variablenadressen und der Manipulation von Zahlenvariablen.

Besonders hervorzuheben sind auch die umfangreichen Befehle zur Dateiverwaltung. Voll integriert ist die Behandlung von relativen Dateien. Es ist sogar möglich, eine Datei bei der Eröffnung beispielsweise als »Nur-Lese-Datei« zu deklarieren, so daß ein versehentliches Überschreiben von Daten von vorneherein ausgeschlossen ist. Alle übrigen, bereits aus dem C128-Modus bekannten Standardfunktionen, sind ebenfalls fast vollständig enthalten.

Sozusagen Hand in Hand arbeiten hier Compiler und Quellcode, denn im Basic-Text können Befehle eingefügt werden, die die Compilierung direkt beeinflussen. So kann für die Ausgabe eines Listings auf einem Drucker ein Seitenvorschub eingefügt, die Seitenlänge festgelegt oder das Listing von Programmteilen ganz unterdrückt werden. Über eine solche Anweisung ist es möglich, weitere Quellcodes in den bestehenden einzubinden und mitzucompilieren. Einen Debugger können Sie auf dieselbe Art und Weise aktivieren. Um die Aus- und Eingabe während des Compilierens zu manipulieren, beispielsweise um Dateien von anderen Laufwerken einzubinden oder ein Listing auf Drucker zu erzeugen, können dem Compiler beim Aufruf verschiedene Parameter mitgeteilt werden, die dann die gewünschten Effekte erzeugen. Dasselbe gilt auch für den Linker, der nach dem Compilieren noch die nötigen Module für ein ablauffähiges COM-File anfügt. Für diese Module existiert auf der Diskette ein Programm, mit dem man selbsterstellte Module für den Einsatz mit dem Linker aufbereiten kann (LIB). Dadurch hat man also die Möglichkeit, den Compiler individuell umzugestalten.

Nach den doch etwas langwierigen Prozeduren des Compilierens und Linkens ist die Ausführungszeit des fertigen Programms geradezu herzerfrischend. Getestet wurde dies mit einem Programm, das einen Zähler von 1 bis 10000 hochzählt. Das Compilieren und Linken dauerte zusammen zirka drei Minuten, während die Ausführungszeit bei 0,5 Sekunden liegt.

Als bestens geeignet erweist sich der CBasic-Compiler für den Anwendungsprogrammierer, der sich mehr in Richtung kommerzieller Anwendung, wie etwa dem kaufmännischen Bereich, bewegt. Die Möglichkeit der strukturierten Programmierung mit Hilfe von Funktionen und Schleifen stellt selbst die Ansprüche von professionellen Programmierern zufrieden. Der große Befehlssatz, das zum Großteil ausführliche, aber nur in Englisch vorliegende Handbuch und der respektable Preis von 174 Mark machen den CBasic-Compiler zu einem unersetzlichen Helfer für den CP/M-Basic-Programmierer. (Guido Weckwerth/bj/rf)

Info: MS-Basic (Microsoft), Vertrieb: Markt&Technik Verlag, Hans-Pinsel-Str. 2, 8013 Haar bei München, Preis: 199 Mark
CBasic (Digital Research), Vertrieb: Markt&Technik Verlag, Hans-Pinsel-Str. 2, 8013 Haar bei München, Preis: 174 Mark



Textverarbeitung mit Protext für den C128 PC

Die Autoren Rudolf Schineis und Hans-Jürgen Thies, beide selbst aktive Programmierer, haben in dem soeben erschienenen Werk ihr Konzept von einer Einführung für Textverarbeitungsneulinge bis hin zu Tips und Tricks für den Profi konsequent durchgehalten. Auch der fortgeschrittene Anwender findet viel Information über die Feinheiten beim Umgang mit Protext 128. In dieser Manier werden zuerst die benötigten Systemkomponenten, die Protext 128 voraussetzt, vorgestellt und die ersten Schritte in dieses hervorragende Textverarbeitungsprogramm unternommen, zum Beispiel das Erstellen einer Arbeitsdiskette. In den folgenden beiden Kapiteln lernt der frischgebackene Anwender von Protext 128 alle Textfunktionen, Korrekturmöglichkeiten und vieles mehr kennen. Am Ende des vierten Kapitels ist der Textverarbeitungsneuling in der Lage, einen perfekten Brief nach eigenen Wünschen zu gestalten. Die folgenden Kapitel wenden sich an den (nun) Fortgeschrittenen: Serienbriefe, trickreiche Anschriftenverwaltung, Module, Jobs, Phrasen und Makros. Hier wird gezeigt, wie leistungsfähig mit einem ausgereiften Textverarbeitungssystem gearbeitet werden kann, ohne daß die Tätigkeiten am C128 zur tatsächlichen Arbeit werden. In der zweiten Hälfte dieses Buches werden wichtige Gebiete, wie etwa der Terminal-Modus in Protext 128, Datenaustausch mit anderen Programmen, alles um den Druckertreiber sowie Sonderfunktionen des Programms vorgestellt, die weit über die Textverarbeitungsnorm hinausgehen. Als sehr hilfreich bei der Arbeit mit Protext 128 stellt sich die alphabetische Befehlsübersicht im Anhang C heraus. Die beiden größten Pluspunkte sammelt dieses Werk sowohl durch die beiliegende Diskette mit Demotexten und Druckertreibern (zum Beispiel für MPS 801, MPS 803, NEC, TA-Gabriele 9009, Epson FX 85) sowie dem Komplettpreis für Buch und Diskette von sage und schreibe 39 Mark.



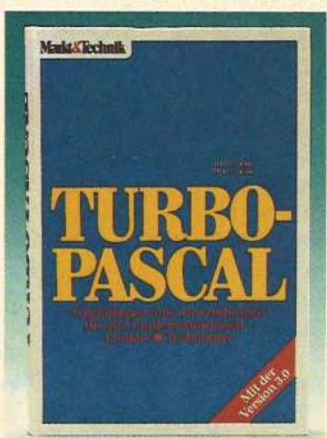
Fazit: Sehr empfehlenswert für alle Protext-128-Anwender.

(E. Schwab/bj)

Info: R. Schineis/H.J. Thies, Textverarbeitung mit Protext für den C128 PC, Markt&Technik Verlag, 258 Seiten, ISBN 3-89090-375-4, Preis: 39 Mark einschließlich Zusatzdiskette

Turbo-Pascal

Die Autoren Irene und Peter Lüke legten in diesem Buch besonderen Wert darauf, den Leser ohne besondere Vorkenntnisse in Turbo-Pascal einzuführen. Dabei werden alle Turbo-Pascal-Versionen bis 3.0 unter CP/M und unter MS-DOS berücksichtigt. Die Sprachelemente wie Variablentypen, Funktionen und Prozeduren, die in Turbo-Pascal bereits definiert sind, werden anhand von Beispielen vorgestellt und in Abfolge ihrer Verwendung beschrieben. Als Ergebnis des Lernteils dieses Buches erhält der Anwender ein Farbkombinationsspiel nach Art von »Master-Mind«, ausführliche Kenntnisse über dessen Aufbau, die Möglichkeiten der strukturierten Programmierung in Turbo-Pascal sowie über den Umgang mit den Compiler-Funktionen überhaupt. Taucht etwa im Spiel ein Variablentyp auf, so werden die Variablenarten allgemein besprochen. Weitere Kapitel des Buches sind beispielsweise den Ein- und Ausgabefunktionen, Strukturanweisungen (begin/end, do/until etc.), Dateiverwaltung (Records), Mengentypen (Sets) und Texttypen (Files) gewidmet. Weiterhin wird ausführlich auf die Turbo-Tools (etwa Turbo-Grafik) und die Funktionen des Turbo-Pascal-Editors eingegangen. Bedauerlicherweise wurde keine alphabetisch geordnete und dokumentierte Befehlsübersicht in dieses Buch aufgenommen, wodurch sich der Wert als Nachschlagewerk reduziert. Insgesamt kann dieses Buch jedoch jedem Einsteiger in Turbo-Pascal empfohlen werden.



(K. Steinmetz/rf/bj)

Info: Irene und Peter Lüke, Turbo-Pascal, Markt&Technik Verlag, 290 Seiten, ISBN 3-89090-150-6, Preis: 49 Mark

Der Einstieg in C

Keinerlei Grundwissen setzt Paul M. Chirlian beim Leser voraus. Am Anfang des Buches werden sogar noch grundlegende Dinge zum Thema Computer angesprochen. Über viele Beispiele wird der Lernende an die Programmiersprache »C« herangeführt und lernt nach und nach alle Elemente dieser zwar leistungsfähigen, aber komplizierten Sprache kennen. Gegen Ende dieses aufschlußreichen Werkes wird das schwierigste »C«-Thema abgehandelt: die Arrays und Pointer. Alle möglichen Variationen werden ausführlich und durch Beispiele unterlegt beschrieben. Zu guter Letzt wird dem Leser noch die Arbeit mit Strings und Strukturen sowie verschiedene Ein-/Ausgabeprozeduren erklärt.



»Der Einstieg in C« erweist sich vor allem für den Anfänger, der keine Kenntnisse in der »C«-Programmierung besitzt, als unentbehrliches Einsteiger- und Nachschlagewerk.

(L. Hartmann/rf/bj)

Info: Paul M. Chirlian, Der Einstieg in C, Markt&Technik Verlag, 290 Seiten, ISBN 3-89090-086-0, Preis: 60 Mark

Wie arbeite ich mit dem Commodore 128

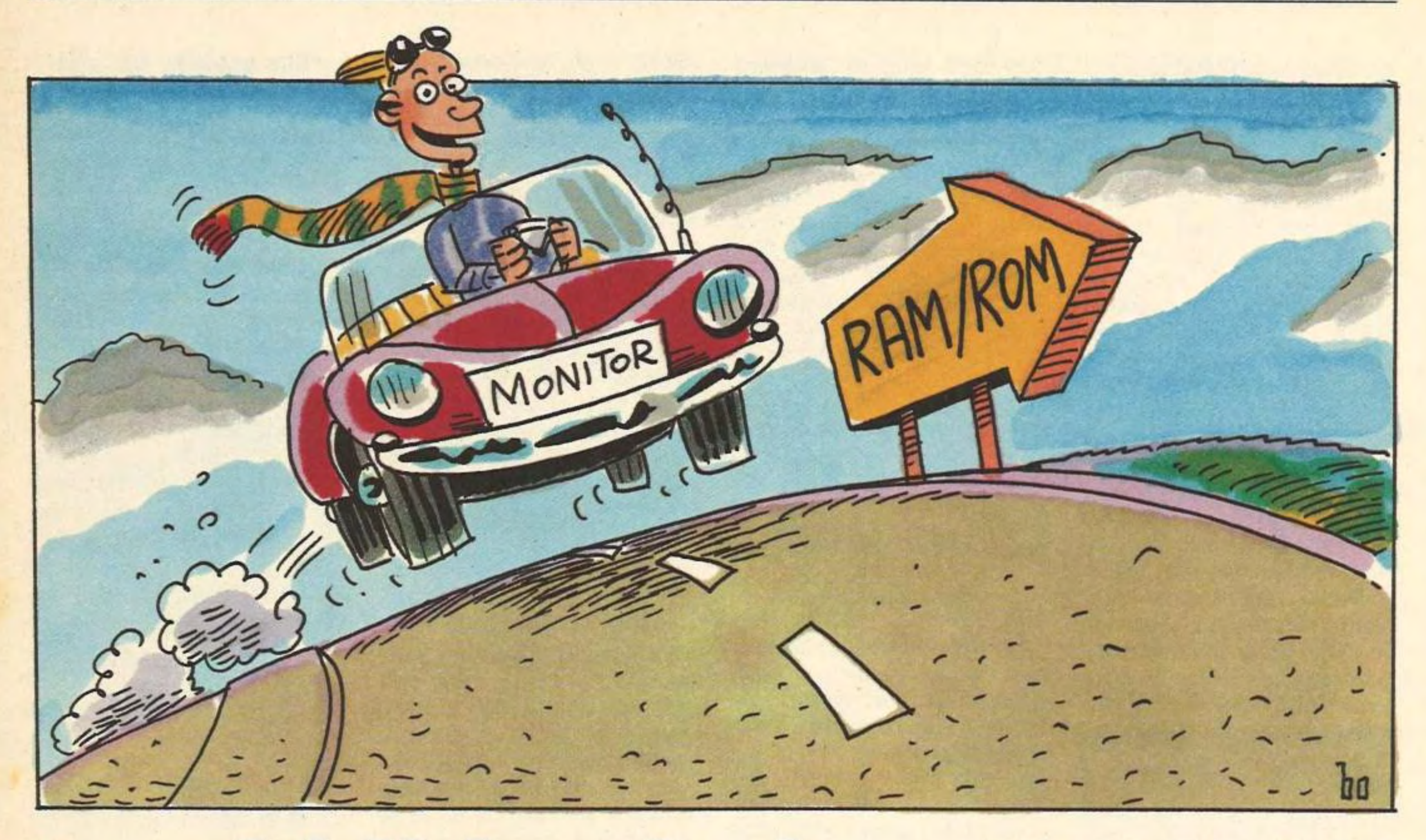
Unter dieser Fragestellung geht der Autor Wolfgang Schneider an den Themenkomplex C128 heran. Das gut durchdachte Konzept konzentriert das Wesentliche zu dem jeweiligen Thema, ohne sich von der Informationsflut der dargestellten Materie überrollen zu lassen oder in die häufig verwendete Fachsprache zu verfallen. Das Buch verbindet die Einführung in das neue Medium C128 mit komprimierter, oft tabellarischer Information, die dennoch anspricht. Weiterhin werden Hinweise, Leitsätze, Erklärungen und verständliche Definitionen optisch markiert, was das Herauskristallisieren von besonders wichtigen Informationen erleichtert. Praktisch alle Gebiete, die unter der obigen Zielsetzung relevant sind, werden behandelt: Es fehlen weder die drei verschiedenen Betriebsarten des C128, allgemeine Überblicke über Programmierung, Hardware des C128 und dessen möglicher Peripherie, noch das Arbeiten mit Basic-Programmen, Kassettenrecorder oder Floppy-Laufwerken. Auch Grafik, Musik und Sprachumfang des C128 anhand kleiner Beispiele sind Teilgebiete des Buches.



Fazit: Wer gerne mehr über seinen C128 wissen möchte und die Informationen verständlich, aber komprimiert bevorzugt, dem sei dieses Werk wärmstens empfohlen. Nachdem man sich das Hintergrundwissen angeeignet hat, kann man immer noch auf fachspezifische Bücher umsteigen, die dann die entsprechende Materie abschließend behandeln.

(J. Rieder/bj)

Info: Wolfgang Schneider, Wie arbeite ich mit dem Commodore 128?, Vieweg-Verlag, 342 Seiten, ISBN 3-528-04472-1, Preis: 48 Mark



Durchblick mit dem TEDMON

Erfreulicherweise steht dem C128-Besitzer mit dem integrierten Monitor TEDMON ein leistungsfähiges Werkzeug zur Verfügung. Obwohl in der Anleitung immer von einem Maschinensprache-Monitor die Rede ist, kann auch der Basic-Programmierer den TEDMON effektiv einsetzen. Der folgende Kurs hilft ihm dabei.

Fast jeder etwas fortgeschrittene Basic-Programmierer stand schon einmal staunend vor dem seltsamen Zahlendschungel, der von einem Maschinensprache-Monitor erzeugt wird und wußte nicht, was er damit anfangen sollte. In diesem Kurs werden Sie lernen, daß der Monitor ein durchaus leistungsfähiges Werkzeug darstellt, das auch der Basic-Programmierer hervorragend einsetzen kann, wenn er weiß, wie der Monitor bedient werden muß.

Natürlich geht dies alles nicht ohne Theorie, aber der Kurs ist so gestaltet, daß Sie alles gleich an Ihrem C128 testen können. Dazu sollten Sie Basic-Kenntnisse mitbringen; wenn Sie gar Maschinensprache beherrschen, wird Ihnen der Kurs noch mehr helfen, erforderlich ist das jedoch nicht.

Zur Funktion des Monitors: Zunächst einmal dient der Monitor nur zum Zugriff auf den Speicher des Computers. Dies klingt nicht sehr interessant, aber man muß sich nur vor Augen führen, wie wichtig der Speicher für unseren Computer ist.

Aufgrund der Befehlsstärke des Basic 7.0 kann man fast vergessen, daß auch ein Basic-Programm nichts anderes als eine Ansammlung von Speicherinhalten ist. Über Basic greift man mittels POKE, PEEK und WAIT auf den Speicher zu. Speicherzellen kann man als Fächer verstehen, in denen Informationen - die Speicherinhalte - untergebracht sind. Die folgenden Erläuterungen von Grundbegriffen mögen dem Fortgeschrittenen langweilig erscheinen, für den Anfänger sind sie unentbehrlich.

Die Chips (zum Beispiel Sound-Chip SID 6581) und das Betriebssystem (das Programm, das den Computer vom Einschalten an dauernd steuert) benötigen viele Zugriffe auf den Speicher. Wenn sich der Computer irgend etwas merken will (zum Beispiel eine Zahl, einen Basic-Befehl oder einen String), so muß er es im Speicher - seinem Gedächtnis - ablegen.

Der Speicher ist in Speicherzellen (auch Speicherplätze genannt) unterteilt. Jeder Speicherplatz kann eine Zahl von 0 (minimal) bis 255 (maximal) oder einfach ein Zeichen (als ASCII-Code, zum Beispiel 65 für »A«) aufnehmen. Diese Speicherkapazität (ein Zeichen) heißt ein Byte. Haben Sie eine Einheit von 1024 Byte, so spricht man auch von einem KiloByte oder kurz KByte. Weitere Unterteilungen besprechen wir beim Binärsystem.

Jede Speicherzelle ist durch eine Adresse genau bestimmt. Die Adresse ist eine Zahl von 0 (minimal) bis 65535 (maximal). Beim C128 kommt noch die Speicherbank hinzu, da wir die 128 KByte mit den Zahlen 0 bis 65535 nicht eindeutig beschreiben könnten. Auch damit werden wir uns noch auseinandersetzen.

In unserem Computer gibt es nun Speicherplätze, die Auskünfte über das Basic-Programm, Grafik-Daten, das Basic-Programm selbst und seine Variablen oder ähnliches enthalten. Mit dem Monitor können wir diese Speicherzellen auslesen (die Inhalte der Speicherzellen anzeigen lassen) und zurückschreiben (die Inhalte ändern und in die Speicherzelle aufnehmen). Dadurch werden uns Einflußnahmen auf den Computer möglich, die uns vorher verschlossen waren.

Mit dem Monitor unterhalten wir uns ähnlich wie mit dem Basic 7.0 im Direktmodus: Wir geben Kommandos (meist mit Parametern) ein, die dann ausgeführt oder mit einer Fehlermeldung quittiert werden. Damit der Computer weiß, daß die

Befehle an den Monitor gerichtet sind, müssen wir den Monitor starten. Dies geschieht am einfachsten über den Basic-Befehl MONITOR, der erfreulicherweise schon als Belegung der Funktionstaste <F8> (<SHIFT+F7>) vorgesehen ist.

Wenn wir MONITOR eingeben (oder einfach <F8> drücken), erhalten wir die Meldung »MONITOR«.

Darunter steht die zweizeilige Register-Anzeige, die wir zunächst ignorieren wollen (sie ist ohnehin nur für Maschinensprache-Programmierer interessant).

Diese Meldung »MONITOR« entspricht der Meldung »COMMODORE BASIC V7.0...«, die beim Reset ausgegeben wird.

Wenn nun der Cursor blinkt, ist er – wie im Basic – das Signal, daß der Computer auf weitere Eingaben wartet.

Um noch einmal auf die Meldung »MONITOR« zurückzukommen: Diese Meldung signalisiert auch, daß – solange wir uns im Monitor befinden – sämtliche Basic-Befehle außer Kraft sind. Von nun an werden nur noch die speziellen Monitor-Kommandos akzeptiert, von denen wir schon im nächsten Abschnitt den ersten kennenlernen werden. Diese Monitor-Kommandos können übrigens – wie Basic-Kommandos – über <RUN/STOP> abgebrochen werden.

In Bild 1 sehen Sie noch einmal, wie ein Bildschirm beim Starten des Monitors <F8> aussieht.

Der erste Befehl: X

Wie gesagt, im Monitor gelten die Basic-Befehle nicht mehr. Wenn man wieder in den Eingabemodus des Basic-Interpreters zurück möchte (vielleicht verspüren Sie so etwas wie Heimweh...), ist dafür der einfachste Monitor-Befehl vorgesehen: Sie geben einfach X ein und drücken – wie in Basic – zur Bestätigung <RETURN>. An der Meldung »READY.« können Sie erkennen, daß Sie sich wieder im vertrauten Basic befinden. X hat keine Parameter, das heißt Sie müssen keine weiteren Angaben machen. Dieser Befehl entspricht einem END in Basic.

Hinweis: Der Befehl, der auf den Basic-Befehl MONITOR folgt, wird nach X nicht abgearbeitet!

In Bild 2 finden Sie alle Informationen zu unserem ersten Befehl. Diese Kurzübersichten haben folgendes Schema:
- Befehl (Buchstabe des Befehls)
- Syntax (wie werden Parameter übergeben)
- Wirkung
- englisches Wort, von dem die Ein-Buchstaben-Abkürzung abgeleitet ist

Im Fall des Befehls X ist dies nicht sehr umfangreich, da der Befehl, wie gesagt, keine Parameter kennt.

```
Monitor ← Der Befehl zum Starten (über F8 erreichbar)

Monitor ←Einschaltmeldung

     PC   SR AC XR YR SP  }
  ; FB000 00 00 00 00 F8  } ← Registeranzeige

■

↑
Cursor als Aufforderung zur Eingabe von Befehlen
```

Bild 1. Das erscheint beim Starten des Monitors

Befehl:	X
Syntax:	X
	keine Parameter)
Wirkung:	Verlassen des Monitors
englisches Wort:	exit (deutsch verlassen)

Bild 2. Kurzübersicht zum Befehl X

Das Hexadezimal- und das Binärsystem

Nachdem wir so schnell den Einstieg in die Praxis gefunden haben, müssen wir wieder zur Theorie. X ist nämlich der einfachste aller Befehle (weshalb wir ihn zuerst besprochen haben), für etwas kompliziertere Anwendungen fehlen uns nur noch einige Grundkenntnisse und -begriffe.

Wahrscheinlich haben Sie schon gehört, daß das Zahlensystem, mit dem der Computer intern arbeitet, anstelle der Basis 10 (unser gebräuchliches Dezimalsystem) die Basis 2 hat. Die Ziffern sind also nicht 0 bis 9, sondern 0 und 1. 0 wird in bezug auf elektronische Signale als »AUS«-, 1 als »EIN«-Zustand verstanden. Wie wir eine Dezimalzahl in Potenzen der Basis 10 zerlegen können ($1986 = 1*10^3 + 9*10^2 + 8*10^1 + 6*10^0 = 1*1000 + 9*100 + 8*10 + 6*1 = 1000 + 900 + 80 + 6 = 1986$ q.e.d.), können wir auch so den Wert einer Binärzahl ins dezimale Format umwandeln:

$$\%10101101 = 1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$$
$$= 2^7 + 2^5 + 2^3 + 2^2 + 2^0$$
$$= 128 + 32 + 8 + 4 + 1$$
$$= 173$$

Wie Sie sehen, wird einer binären Zahlenangabe das Prozentzeichen »%« vorangestellt, damit wir binäre von dezimalen Zahlen unterscheiden können.

Eine Binärziffer (englisch: binary digit) heißt Bit. Bit ist die Untereinheit von Byte; mit 8 Bit (siehe Beispiel) kann man genau ein Byte darstellen ($0 <= 173 <= 255$).

Um Werte bis 65535 ($65535 = 2^{16} - 1$) darzustellen, brauchen wir genau 16 Bit = 2 Byte. Das eine Byte enthält den niederwertigen Teil (Low-Byte, kurz LB), das andere den höherwertigen (High-Byte, kurz HB). Die gesamte Zahl errechnet sich folgendermaßen:

$$Zahl = LB + 2^8 * HB = LB + 256 * HB$$

Die Bits des HB sind die höchstwertigen acht Bit der gesamten Zahl, die Bits des LB die acht niederwertigsten.

Im übrigen können wir auch im Dezimalsystem ähnliche Zerlegungen, die uns für die Arbeit am Computer allerdings nichts bringen, durchführen:

$$1986 = 86 + 10^2 * 19 = 86 + 100 * 19 = 86 + 1900$$

An unserem Beispiel der Binärdarstellung von 173 läßt sich auch der große Nachteil des Binärsystems für den Menschen aufzeigen: Um eine recht kleine Zahl wie 173 darzustellen, werden viele Ziffern (8!) benötigt; um eine Adresse (0 bis 65535) darzustellen, brauchen wir 16 Ziffern (LB+HB = 8 Bit + 8 Bit = 16 Bit = 16 Binärziffern):

65535 = %11111111 11111111

In Bild 3 und 4 finden Sie noch einige Informationen zum Binärsystem und zur LB/HB-Darstellung in Form von Beispielen.

Nun gibt es ein Zahlensystem, das einen Kompromiß zwischen Binärsystem (für die Maschine leichtverständlich) und Dezimalsystem (für den Menschen sinnvoll, für die Maschine ungeeignet) bietet. Es hat die Basis $2^4 = 16$ und folgende Ziffern:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Die Buchstaben sind auch Ziffern; dabei entspricht A der dezimalen 10, B der 11, C der 12, D der 13, E der 14 und F der 15.

Mit einer Hex-Ziffer (Hexadezimal kürzt man gerne mit Hex ab) kann man soviel verschiedene Werte wie mit 4 ($16=2^4$)

Binärziffern darstellen, nämlich die Werte 0 bis 15. Dennoch ist eine Umwandlung ins Binärformat und umgekehrt viel einfacher. 1 Byte = 8 Bit = 2*4 Bit stellen wir mit zwei Hex-Ziffern dar, 1 Adresse = 2 Byte = 16 Bit = 4*4 Bit mit vier Hex-Ziffern (zwei Ziffern HB, zwei für LB). Hex-Zahlen werden gekennzeichnet, indem der Zahl das Dollarzeichen »$« vorangestellt wird.

Beispiele:
255 = %11111111 = $FF
15 = %1111 = $F oder $0F
4096 = %1000000000000 = $1000
2459 = %100110010110 = $099B

Nun rechnen wir noch die Beispielzahl $D01A ins Dezimalsystem um:

$D01A = $D * 16^3 + $0 * 16^2 + $1 * 16^1 + $A * 16^0
= $D*4096 + $0*256 + $1*16 + $A*1
= 13*4096 + 0*256 + 1*16 + 10*1
= 53248 + 0 + 16 + 10
= 53248 + 16 + 10
= 53274

Geben Sie »+« und unmittelbar dahinter eine Dezimalzahl ein (am »+« erkennt der Monitor, daß es sich um eine Dezimalzahl handelt) und drücken Sie <RETURN>. Da vor dem »+« kein Monitor-Befehl stand, erkennt er, daß er die Zahl nun in andere Zahlensysteme umzurechnen hat. Sie erhalten von Ihrer Dezimalzahl dann

- die Hexadezimaldarstellung mit vorangestelltem »$«
- wieder die Dezimaldarstellung (vorangestelltes »+«)
- die Oktaldarstellung (??) mit vorangestelltem »&«
- und die Binärdarstellung mit vorangestelltem »%«

Spätestens bei den eingeklammerten Fragezeichen haben Sie sicher gemerkt, daß da etwas noch nicht besprochen wurde: die Oktaldarstellung. Wie aus der Bezeichnung hervorgeht, handelt es sich um das Zahlensystem mit der Basis 8 (2^3). Es ist zwar auch nicht schlecht zum Umgang mit Binärzahlen geeignet (3 Bit werden zusammengefaßt und ergeben eine Oktalziffer), aber dagegen sprechen mehrere Gründe. Erstens, weil wir mehr Stellen benötigen als beim Hexadezimalsystem und zweitens, daß das Oktalsystem als Zahlensystem veralteter 4-Bit-Computer längst unüblich ist. Lassen Sie sich von der oktalen Zahlenangabe mit dem vorangestellten »&« also nicht stören. Dieses System werden Sie nicht brauchen.

Im übrigen können Sie auch umgekehrte Umrechnungen durchführen; Sie müssen nur das richtige Symbol vor die Zahl setzen ($ für hex, % für bin, + für dez, & für okt). Probieren Sie einmal folgende Beispiele durch:

$FF00
+768
%10110111011 eingeben und <RETURN> drücken
$3455
+3455

Beachten Sie den Unterschied des Wertes der beiden untersten Beispielzahlen; $3455 ist eine Hexadezimalzahl mit dem Dezimalwert 13397 und somit größer als die Dezimalzahl +3455.

Weil die Hexadezimalzahlen so wichtig sind (daß Sie auch praktisch sind, wird sich gleich zeigen), können wir sie auch vom Basic aus umrechnen.

Da hexadezimale Zahlen die Ziffern A bis F enthalten, die nicht numerisch im Sinne des Dezimalsystems sind, müssen Hexadezimalzahlen in Form von Strings eingegeben werden (zum Beispiel A$="FFF0"), in denen kein »$« vorangestellt wird. Die Funktion DEC wandelt den Hex-String dann in eine Dezimalzahl um: A=DEC(A$) oder A=DEC("FFF0"). PRINT DEC("FCE2") ergibt beispielsweise 64738.

Die umgekehrte Umrechnung ist besonders nützlich; sie erfolgt über HEX$ ($ deshalb, weil das Ergebnis nur ein String sein kann, siehe oben): PRINT HEX$(64738) ergibt FCE2.

Das Argument von HEX$ muß – im Gegensatz zum Monitor-Befehl $ – zwischen 0 und 65535 liegen. Das sind alle Werte, die sich mit 16 Bit = 2 Byte (LB und HB) = 2*2 Hex-Ziffern = 4 Hex-Ziffern darstellen lassen. Das Ergebnis wird so mit Nullen aufgefüllt, daß es vierstellig ist.

Dies ist übrigens das praktische an den Hex-Zahlen: Um eine Adresse darzustellen, kommt man immer mit vier Stellen aus, während man im Binärsystem bis zu 16 und im Dezimalsystem bis zu fünf benötigt.

Wir können also festhalten, daß für die Angabe von Adressen das Hexadezimalsystem geeigneter ist als das Dezimalsystem.

Binäre Zahl: %1011101110

Ziffer	1	0	1	1	1	0	1	1	1	0
Stelle Nr.	9	8	7	6	5	4	3	2	1	0

%1011101110 =	0 *2↑0	(0 als Ziffer für Stelle 0)
	+1 *2↑1	(1 als Ziffer für Stelle 1)
	+1 *2↑2	(1 als Ziffer für Stelle 2)
	+1 *2↑3	(1 als Ziffer für Stelle 3)
	+0 *2↑4	(0 als Ziffer für Stelle 4)
	+1 *2↑5	(1 als Ziffer für Stelle 5)
	+1 *2↑6	(1 als Ziffer für Stelle 6)
	+1 *2↑7	(1 als Ziffer für Stelle 7)
	+0 *2↑8	(0 als Ziffer für Stelle 8)
	+1 *2↑9	(1 als Ziffer für Stelle 9)

= 0*1 + 1*2 + 1*4 + 1*8 + 0*16 + 1*32 +
1*64 + 1*128 + 0*256 + 1*512
= 0 + 2 + 4 + 8 + 32 + 64 + 128 + 512
= 2 + 4 + 8 + 32 + 64 + 128 + 512
= 750

Bild 3. Beispiel zum Rechnen mit Binärzahlen

Gehen wir von der Binärzahl %1011011100011101 aus.

	High-Byte	Low-Byte
%	10110111	00011101
$	B7	1D
#	183	29

%1011011100011101 = $ B7 1D = 183*256 + 29
↑ ↑ ↑ ↑
HB LB $B7 $1D

Bild 4. Low- und High-Bytes

Der »Grundbefehl« der Monitore: M

Der Befehl M, den wir nun kennenlernen wollen, ist der wichtigste Befehl eines Monitors. Die Abkürzung M wird daher oft als Abkürzung für Monitor verstanden. M kommt aber wohl eher von »memory dump« (Speicherauflistung).



```
MONITOR
    PC   SR AC XR YR SP
; FB000 00 00 00 00 F8

>F4000 4C 23 40 4C 09 40 4C 4D A8 20 CC FF 20 7A 41 20:
>F4010 8D 41 20 12 41 20 38 52 A9 00 85 15 58 4C 37 4D:
>F4020 00 FF FF 20 7A 41 20 51 42 20 45 40 20 9B 41 AD:

DASSELBE MIT 8 BYTES PRO ZEILE

MONITOR
    PC   SR AC XR YR SP
; FB000 00 00 00 00 F8

>F4000 4C 23 40 4C 09 40 4C 4D:
>F4008 A8 20 CC FF 20 7A 41 20:
>F4010 8D 41 20 12 41 20 38 52:
>F4018 A9 00 85 15 58 4C 37 4D:
>F4020 00 FF FF 20 7A 41 20 51:
```

Bild 5. Beispielausdruck des M-Befehls

Dieser Befehl ist bei jedem Monitor vorhanden, heißt fast immer M und hat auch immer ähnliche Syntax. Sogar die Disketten-Monitore, mit ihnen lassen sich Inhalte von Disketten verändern, haben diesen Befehl.

M erzeugt einen sogenannten »Dump« (Auflistung eines Bereichs, ohne bei der Ausgabe zu selektieren, das heißt nur bestimmte Werte auszugeben).

Geben Sie bitte »M F4000 F4020« ein; in Bild 5 und auf Ihrem Bildschirm sehen Sie das Ergebnis. Ein »memory dump« hat folgendes Format:

Basisadresse Hex-Angabe der Speicherinhalte: ASCII-Zeichen

Besprechen wir die einzelnen Angaben:

Das >-Zeichen zeigt an, daß es sich um ein Dump des M-Befehls handelt. Dies ist für das Ändern der Speicherinhalte, was wir noch besprechen werden, elementar.

Die Basisadresse wird – was sofort auffällt – mit fünf Hex-Ziffern angegeben. Die erste Ziffer gibt die Speicherbank an. In diesem Fall ist es $F (dezimal 15), also die Speicherbank des ROM-Bereichs. Die folgenden vier Ziffern sind die gewöhnliche Hex-Darstellung der Adresse. Auch bei der Angabe der Adressen für unser Beispiel haben wir übrigens die fünfstellige Angabe gewählt.

Die Basisadresse gibt an, auf welche Adresse sich die darauffolgenden Angaben beziehen. Der Basisadresse folgen nämlich die Hex-Darstellungen der Inhalte aller Adressen – beginnend mit der Basisadresse.

Aus unserem Beispiel (sehen Sie auf Ihren Bildschirm, wenn Sie den Beispiel-Befehl eingegeben haben, oder auf das bereits erwähnte Bild 5) geht hervor, daß $4C (76) der Speicherinhalt der Adresse $4000 (16384) in Bank $F (15) ist, $23 (35) der von $4001 (16385) in derselben Bank, $40 (64) der von $4002 (16386)... $8D (141) ist – siehe nächste Zeile des Dumps – der Inhalt von $4010 (16400) und so weiter.

Auf die Hex-Bytes folgt der Doppelpunkt »:«. Dieser zeigt an, daß die numerischen Angaben an dieser Stelle aufhören und ist vor allem für das Ändern von Speicherinhalten wichtig. Der Doppelpunkt wird bei einer Eingabe als Endmarkierung einer Anweisung verstanden.

Dem Doppelpunkt folgt die (inverse) ASCII-Darstellung der Speicherinhalte. Falls es sich um Speicherinhalte handelt, die nicht als ASCII-Zeichen (Buchstaben, Ziffern, Grafikzeichen) darstellbar sind, wird ein reverser Punkt ».« ausgegeben. Achtung: Dieser Punkt kann auch für den Code $2E stehen. Es empfiehlt sich also, auch die hexadezimalen Werte zu betrachten.

Die ASCII-Darstellung ist vor allem dann wichtig, wenn Speicherbereiche, in denen ASCII-Codes abgelegt sind, untersucht werden sollen. Geben Sie einmal »M F41BB F424B« ein, und Sie finden rechts die Einschaltmeldung des C 128 und links die Hex-Darstellung der ASCII-Codes.

Hier sei auch darauf hingewiesen, daß der M-Befehl – wie alle Monitor-Kommandos – durch Drücken von <RUN/STOP> abgebrochen werden kann.

Nun wollen wir noch die Syntax von M, die bislang bewußt übergangen wurde, besprechen.

Die normale Syntax ist »M Anfangsadresse Endadresse«, wobei Anfangs- und Endadresse hexadezimal dargestellt werden, solange nichts anderes angegeben wird. Ist die Angabe hexadezimal und hat fünf Stellen (Beispiele: F400C, 10400, 04001), kennzeichnet die höchstwertige Stelle die Speicherbank (0,1,F). Bei Angaben mit weniger hexadezimalen Stellen wird die voreingestellte Bank (0) verwendet.

Als einer der wenigen Monitore überhaupt, bietet TEDMON die Möglichkeit, die Parameter auch in anderen Zahlensystemen zu übergeben. Wichtige Einschränkung: Die Speicherbank kann nicht eingestellt werden, man kann nur mit der voreingestellten Konfiguration (Bank 0) arbeiten. Für Adressen, bei denen die Angabe der Speicherbank überflüssig ist, weil sie in dem Bereich liegen, der in allen Bänken gleich ist ($0000 bis $3FFF oder dezimal 0 bis 16383), kann man damit aber ganz gut leben. Um die Adressen dezimal 700 bis 900 zu dumpen, gibt man einfach »M+700+900« ein. Da die Adressen dezimal 700 bis 900 in diesem gemeinsamen Bereich dezimal 0 bis 16383 liegen, müssen wir nicht die Speicherbank angeben und können auch problemlos die dezimale Darstellung der Parameter einsetzen.

Man kann die Parameter für M teilweise weglassen. »M +700« beispielsweise gibt einen Dump ab 700 aus; es werden zwölf Zeilen (diese Anzahl ist unabhängig von 40- oder 80-Zeichen-Darstellung) ausgegeben. Anschließend lassen sich neue Eingaben machen. Mit M (ohne Parameter!) wird ab der letzten Adresse, die bearbeitet wurde, ein Dump ausgegeben. Dies ist vor allem dann sinnvoll, wenn man die Ausgabe über <RUN/STOP> abgebrochen hat und wieder fortsetzen möchte.

In Bild 6 finden Sie eine Kurzübersicht zum M-Befehl.

Wir wollen uns nun ein wenig mit der Funktionsweise des M-Kommandos beschäftigen.

Der Monitor liest die Speicheradresse ähnlich wie der PEEK-Befehl im Basic. Diese werden zunächst in hexadezimaler Form ausgegeben und dann als ASCII-Zeichen (Steuerzeichen als Punkt). Befinden wir uns in der 40-Zeichen-Darstellung, werden 8 Byte pro Zeile, bei 80-Zeichen-Darstellung 16 Byte, ausgegeben.

Listing 1 ist ein Basic-Programm, das den M-Befehl simuliert. Es ist natürlich keine echte Alternative zum Monitor, der

ja in Assembler geschrieben und folglich viel schneller ist. Am Listing kann man sich aber die Wirkungsweise des M-Befehls gut veranschaulichen. Die Variable ZL bekommt in Zeile 220 zunächst den Wert 8. Der Rest der Zeile addiert noch 8 hinzu, falls die 80-Zeichen-Darstellung eingeschaltet ist. Dies wird anhand der Speicherzelle 215 festgestellt, die dann den Wert 128 beinhaltet, wenn 80 Zeichen pro Zeile eingestellt sind. Das weitere Programm entnimmt aus der Variablen ZL, wieviele Bytes pro Zeile angezeigt werden sollen.

Die Eingaben, die das Programm fordert, müssen mit Ausnahme der Bank, die dezimal angegeben wird, hexadezimal sein (wie die DEC-Befehle in den Zeilen 250/260 zeigen, die die hexadezimale Eingabe in einen dezimalen Wert umwandeln, damit das restliche Basic-Programm damit arbeiten kann).

Nun hätten wir die Funktionsweise und die Syntax des M-Befehls besprochen. Da M unser erster variantenreicher Befehl ist, ging dies noch etwas mühselig; in Zukunft müssen wir aber nicht mehr die Unterbrechungsmöglichkeit durch <RUN/STOP> oder die Darstellung der Parameter in anderen Zahlensystemen und andere grundsätzliche Dinge besprechen. Außerdem sind die anderen Befehle viel einfacher, und oft werden wir uns sehr kurz fassen können. Wenn Sie aber mit dem M-Befehl umgehen können, haben Sie eine große Hürde in der Anwendung des Monitors genommen.

Vielleicht haben Sie sich gefragt, was denn der M-Befehl für einen Nutzen hat. Diese Frage ist durchaus berechtigt, aber wir hoffen, daß die Anwendungsbeispiele, die wir im folgenden durchprobieren, zeigen, daß man mittels M an viele Informationen herankommt.

Beispiel 1:

Lassen Sie doch einmal das Betriebssystem anzeigen (das Maschinenprogramm, das unmittelbar nach dem Einschalten des Computers aktiviert wird und alle softwaremäßigen Vorgänge steuert) oder den Basic-Interpreter. Diese beiden Programme sind fest eingebaut und können mit »M F4000 FF000« auf den Bildschirm gebracht werden. Vor allem die ASCII-Darstellung auf der rechten Seite des Ausdrucks zeigt dann Basic-Befehlswörter, Fehlermeldungen und so weiter.

Befehl:	M
Syntax:	M Anfangsadresse bis Endadresse
Wirkung:	Memory Dump von Anfangsadresse bis Endadresse bei Weglassen der Endadresse: von der Anfangsadresse 12 Zeilen lang bei Weglassen beider Parameter:
englisches Wort:	Memory Dump (Speicherauflistung)

Bild 6. Kurzübersicht zum Befehl M

Beispiel 2:

Im Speicher beinhalten manche Adressen wichtige Werte für den Computer in Form von Zeigern. Ein Zeiger besteht aus zwei Byte, dem Low- und High-Byte einer Adresse.

Im Zeiger $2D/$2E ($2D enthält das LB, $2E das HB) ist die Anfangsadresse des aktuellen Basic-Programms gespeichert. Geben wir also »M 2D 2E« ein. Da der Monitor immer nur ganze Zeilen anzeigt, erhalten wir nicht nur die beiden Werte aus $2D und $2E, sondern auch weitere Zahlen, die uns aber an dieser Stelle nicht interessieren.

Die beiden am weitesten links stehenden Hex-Bytes sind natürlich die Inhalte von $2D und $2E, wie wir der Basisadresse ($0002D=$2D in Bank $0) entnehmen können. In der Regel erhalten wir $01 für $2D. $01 ist also das LB der Adresse, ab der das Basic-Programm beginnt. Der zweite Wert ist das HB; hierfür erhalten wir direkt nach dem Einschalten $1C. Falls wir die hochauflösende Grafik eingeschaltet haben, ist es $40. Die Adresse ist also $1C01 beziehungsweise $4001, falls der Bereich für hochauflösende Grafik reserviert ist.

Mit Hilfe des $-Befehls zur Umrechnung einer Hex-Zahl ins Dezimalformat können wir noch die dezimalen Werte berechnen lassen. Zur Überprüfung des Ergebnisses können Sie mit X ins Basic zurückgehen und dort über »PRINT PEEK(45)+256*PEEK(46)« das Ergebnis prüfen, das vom genannten Basic-Ausdruck allerdings in dezimaler Form ausgegeben wird.

Die Endadresse des Basic-Programms ist immer in $1210/$1211 (dezimal 4624/4625) gespeichert. Mit dem Monitor ist also »M 1210 1211« einzugeben. Die weitere Behandlung entspricht dem Auslesen des Zeigers ($2D/$2E) auf den Basic-Anfang.

Vom Basic aus können Sie Ihr Ergebnis diesmal mittels »PRINT PEEK(4624)+256*PEEK(4625)« prüfen.

In den Ausdrücken zur Berechnung der Zeigerinhalte können Sie übrigens gut die Formel Wert = LB + 256*HB« erkennen. Bei der Berechnung über den Monitor-Befehl $ sehen Sie, daß man im Hexadezimalsystem nur die Stellen des HB als erste zwei Stellen, die des LB als letzte zwei Stellen einsetzen muß, um den gesamten Wert zu erhalten.

Beispiele: LB=$12,HB=$34 = Wert = $3412
LB=$E2,HB=$03 = Wert = $03E2

Dieses Verfahren, die Stellen des HB als höherwertige Stellen einzusetzen, ist mathematisch völlig korrekt; ähnlich geht man vor, wenn man im 10er-System bei der Multiplikation einer Zahl mit einer Zehnerpotenz die Zahl als linke Stellen, die Nullen als rechte Stellen verwendet: 54*100 = 54 00 =5400

Abschließend zum M-Befehl (dieses Kommando werden wir nun dauernd verwenden) noch zwei Hinweise.

Schalten Sie einmal auf die 40-Zeichen-Darstellung um und geben »M+1024+2023« ein. Was auf dem Bildschirm erscheint, ist nichts anderes als der Inhalt des 40-Zeichen-Bildschirmspeichers. Ein Auslesen des 80-Zeichen-Bildschirmspeichers ist mit dem Monitor leider nicht möglich.

```
100 REM ******************************
110 REM *                            *
120 REM * SIMULATION DES MONITORBE-  *
130 REM *                            *
140 REM *  FEHLS 'M' MITTELS BASIC   *
150 REM *                            *
160 REM ******************************
170 REM *                            *
180 REM *  1986 BY FLORIAN MUELLER   *
190 REM *                            *
200 REM ******************************
210 :
220 ZL=8 - 8*(PEEK(215)=128): REM ANZAHL DER ZEI
    CHEN PRO ZEILE (8 ODER 16) BESTIMMEN
230 PRINT CHR$(13) CHR$(13)"MEMORY DUMP MITTELS
    BASIC{DOWN}" CHR$(7)
240 INPUT "SPEICHERBANK (0,1,2,3 ODER 15)";BA
250 INPUT "ANFANGSADRESSE";AN$: AN=DEC(AN$)
260 INPUT "ENDADRESSE{4SPACE}";EN$: EN=DEC(EN$)
270 PRINT CHR$(13)"MEMORY DUMP $";AN$;" - $";EN$
    ;" IN BANK"BA
280 AD=AN: BANK BA
290 DO UNTIL AD>EN
300 : PRINT CHR$(13)">";RIGHT$(HEX$(BA),1);HEX$(
    AD);
310 : FOR L=1 TO ZL
320 : : PRINT " ";RIGHT$(HEX$(PEEK(AD)),2);: AD=
    AD+1
330 : NEXT L
340 : PRINT ":" CHR$(18);
350 : FOR L=AD-ZL TO AD-1
360 : : I=PEEK(L)
370 : : IF I<32 OR (I>127 AND I<160) THEN PRINT
    ".";: ELSE PRINT CHR$(I);
380 : NEXT L
390 LOOP
400 RUN
```

Listing 1. Simulation des M-Befehls in Basic


64ER ONLINE
64er-online.de
64er-online.net

Zweiter Hinweis: Wenn das Basic-Programm in Form eines Hex-Dumps ausgegeben werden soll, können Sie die Parameter (Anfangs- und Endadresse) so herausfinden, wie wir es schon beim Auslesen der entsprechenden Zeiger getan haben. Sie werden fast das ganze Programm bei der ASCII-Darstellung wiederfinden, aber kein Befehlswort. Diese werden nämlich als 1-Byte-Werte (sogenannte Token, mehr darüber später) im Speicher abgelegt, die dann meist als Grafikzeichen erscheinen. Der Basic-Befehl LIST muß diese Token erst mit Hilfe einer Berechnungsroutine und einer Umwandlungstabelle in Klartext - die Befehlswörter - umwandeln, der bei LIST ausgegeben wird. Der Grund für diese Prozedur ist, daß 1-Byte-Werte leichter und schneller bearbeitet werden können als mehrere Bytes umfassende Befehlswörter.

Das Ändern von Speicherzellen mit Memory Dump

Im letzten Abschnitt wurde Ihnen für die Anzeige von > am Anfang jeder Dump-Zeile nur die Erklärung gegeben, daß dieses Zeichen es zuläßt, die Speicherinhalte der angezeigten Adressen zu ändern.

Geben Sie bitte »M 1008« ein; Sie werden sehen, daß in diesem Bereich die Belegungen der Funktionstasten gespeichert sind. Wenn Sie nun mit dem Monitor die angezeigten Speicherinhalte ändern wollen, geht dies ganz einfach: Gehen Sie mit dem Cursor auf die Stelle, die geändert werden soll, und führen Sie die Änderung durch. Sie können sowohl die Adresse als auch die Hex-Bytes ändern. Das Ändern der ASCII-Codes wird ignoriert. Drücken Sie dann <RETURN> zur Bestätigung, und die Änderungen werden übernommen. Die Zeile wird noch einmal ausgegeben, damit auch die ASCII-Codes an die neuen Speicherinhalte angepaßt werden.

Daß auf so einfache Weise ein Speicherinhalt geändert werden kann, ist darin begründet, daß das >-Zeichen am Anfang der Zeile ein gültiger Monitor-Befehl ist.

Seine Syntax ist > Basisadresse Speicherinhalte, zum Beispiel > 100A 42 45 49 53 50 (siehe Bild 7).

Wenn Sie jedoch die Anzeige unseres Beispiels »M F4000 F4020« ändern wollen, erkennen Sie an der nochmaligen Anzeige der Zeile, daß die Änderungen nicht übernommen wurden. Die Erklärung ist ganz einfach: Es handelt sich um einen Speicher, der nicht beschreibbar ist. Da fast alle Speicherzellen in Bank 15 ($F) ROM-Speicherzellen sind, kann man die Adressen $4000 bis $4020 in Bank $F nicht ändern, denn die grundlegende Eigenschaft von ROM-Speichern ist die, daß seine Inhalte nicht geändert werden können.

Die Abkürzung ROM ist Ihnen unter Umständen noch nicht bekannt, weshalb wir hier die Begriffe ROM und RAM besprechen wollen.

Der Großteil des Speichers unseres C128 ist RAM (Random Access Memory). Random Access Memory heißt, daß es sich um Speicher handelt, auf den in beliebiger Form zugegriffen werden kann. RAM-Speicherzellen können also sowohl beschrieben als auch ausgelesen werden. Daher übersetzt man RAM mit »Schreib/Lese-Speicher«.

Befehl:	>
Syntax:	>Basisadresse Byte1 Byte2 ... ByteX
Wirkung:	Im Speicher werden ab der Basisadresse die der Basisadresse folgenden Bytes abgelegt.
englisches Wort:	-

Bild 7. Kurzübersicht zum Befehl >

Der Nachteil des RAM-Speichers ist, daß er beim Ausschalten des Computers gelöscht wird. Darum ist der RAM-Speicher für ein Betriebssystem, das man ja immer benötigt, ungeeignet.

ROM-Speicher (Read Only Memory) kann - wie die englische Bezeichnung »read only« sagt, nur gelesen werden, Schreibzugriffe sind dagegen nicht möglich. Der große Vorteil ist, daß die Daten im ROM beim Ausschalten nicht verloren gehen.

Das Programm, auf das der Computer immer wieder zugreift, nämlich das Betriebssystem, ist im ROM gespeichert.

Kurz: Es besteht keine Möglichkeit, bestimmte Speicherzellen zu ändern; dies wäre allerdings auch nicht sinnvoll.

Das Überschreiben von RAM-Zellen ist eine Anwendung, die wir noch etwas genauer besprechen wollen, zum Beispiel das Ändern von ASCII-Tabellen (also reiner Text) in Maschinenprogrammen.

Beim Ändern von solchen Maschinenprogrammen (genauer gesagt: der ASCII-Tabellen solcher Maschinenprogramme) geht man in drei Schritten vor:
1. Programm mittels DLOAD oder BLOAD laden
2. Vom Monitor aus Änderungen durchführen
3. Monitor verlassen (X) und das Programm mittels DSAVE oder BSAVE speichern.

Uns wird hier natürlich der zweite Schritt am meisten interessieren, zunächst aber wollen wir klären, welche Programme wir bei unserem jetzigen Kenntnisstand bearbeiten können und woran man ein Maschinenprogramm erkennt.

Vorher noch zwei wichtige Hinweise. Erstens: Sie benötigen, wie gesagt, keine Maschinensprache-Kenntnisse, es reicht, wenn Sie diesen Kurs bis zu dieser Stelle verfolgt haben. Zweitens: Natürlich kann man auch Basic-Programme mit dem Monitor ändern, doch gibt es dazu ja den Basic-Editor, der viel komfortabler ist. Lediglich für spezielle Änderungen an Basic-Programmen benötigt man den Monitor, und das werden wir an späterer Stelle behandeln.

Die Maschinenprogramme, die wir schon manipulieren können, müssen bestimmte Bedingungen erfüllen:
1. Das Programm muß mindestens aus einer Basic-Zeile bestehen, die beim LISTen erscheint (zum Beispiel eine Zeile »1986 SYS 7183« oder »1986 SYS PEEK(45)+256*PEEK(46)+350«).
2. Das Programm muß mit dem Basic-Befehl RUN gestartet werden können (zum Beispiel mit RUN "FILENAME" oder DLOAD"FILENAME" und anschließendem RUN).
3. Das Programm muß mit DLOAD ladbar sein (und zwar so, daß man es danach mit RUN starten kann).
4. Das Programm muß nach dem Laden über DLOAD auch mit DSAVE gespeichert werden können.

Laden Sie nun bitte ein beliebiges Programm, das Sie manipulieren wollen.

Springen Sie dann mit <F8> in den Monitor und dumpen Sie ab der Anfangsadresse des Basic-Programms (in der Regel $1C01) bis zur Endadresse. Wie man Anfangs- und Endadresse eines Basic-Programms im Speicher mit dem M-Befehl feststellt, hat der letzte Abschnitt gezeigt.

Wenn Sie beim Dump eine Stelle finden, an der interessante ASCII-Codes abgelegt sind, drücken Sie zur Unterbrechung des Dumps einfach <RUN/STOP>. Nehmen wir an, Sie haben folgende Zeile entdeckt:

>01F85 28 43 29 20 31 39 38 36 20 42 59 20 58 59 5A 0D:(C) 1986 BY XYZ.

Die ASCII-Darstellung, rechts vom Doppelpunkt, erscheint bei Ihnen natürlich invers. Außerdem werden Sie nach »M 1F85« diese fiktive Zeile nicht erhalten.

Diese Zeile soll uns nur als ausführliches Beispiel dienen, wie man Änderungen durchführt.

Wenn Sie das Beispiel verfolgen wollen, geben Sie einfach die oben stehende Zeile bis zum Doppelpunkt so ein, wie Sie dort steht; dann speichern Sie aber ein eventuell vorher geladenes Programm nach den Änderungen nicht, denn Sie haben es unter Umständen durch diese Zeile zerstört!

Nun wollen wir an unserer Beispielzeile eine kleine Textänderung durchführen, weil uns das Copyright stört. Unser neuer Text soll nun »NO COPYRIGHT!!!« lauten. Das letzte Byte ($0D=13) soll bestehen bleiben, da es sich um ein wichtiges Steuerzeichen handelt.

Um die Änderung durchzuführen, fahren wir mit dem Cursor auf die erste 2 (im Hex-Byte $28 am Anfang der Zeile) und überschreiben jetzt 15 Hex-Byte mit den ASCII-Codes unserer neuen Meldung:

4E 4F 20 43 4F 50 59 52 49 47 48 54 21 21 21

Nach der Bestätigung durch <RETURN> wird die Zeile neu aufgebaut und wir sehen nun folgendes Bild:

>01F85 4E 4F 20 43 4F 50 59 52 49 47 48 54 21 21 21 0D:NO COPYRIGHT!!!.

An den ASCII-Codes können wir sofort und bequem erkennen, daß die Änderung korrekt durchgeführt wurde.

Nach dieser großen Manipulation fällt uns ein, daß der Text so viel schöner wäre: »NO COPYRIGHTS .«, wobei der Punkt am Ende des Textes wirklich erscheinen soll und kein Steuerzeichen ist!

Diesmal müssen wir nicht gleich eine ganze Zeile ändern, es reicht, an den letzten Bytes folgende Änderungen durchzuführen:

1. das erste $21-Byte soll zum »S« werden
2. das zweite $21-Byte soll zum Leerzeichen » « werden
3. das dritte $21-Byte soll zum Punkt ».« werden

Da wir nur drei von 16 Byte ändern müssen, lohnt es nicht, die ganze Zeile neu zu schreiben. Wir fahren nur mit dem Cursor auf das erste $21-Byte, überschreiben es mit »53«, dem Code für »S«, (das war Änderung 1, siehe oben). Dann fahren wir mit dem Cursor auf das zweite $21-Byte auf die Ziffer 1 und überschreiben sie mit »0«, wir erhalten $20, den Code oder ein Leerzeichen (das war Änderung 2, siehe oben). Zu guter Letzt fahren wir auf die Ziffer 1 des nächsten $21-Bytes und überschreiben es mit »E«, so daß sich der Punkt-Code $2E ergibt. Nach diesen Änderungen drücken wir <RETURN> und erhalten folgende Zeile:

>01F85 4E 4F 20 43 4F 50 59 52 49 47 48 54 53 20 2E 0D:NO COPYRIGHTS ..

Wir sehen auch hier, daß die Änderung übernommen wurde. Aber halt, da soll doch nur ein Punkt ausgegeben werden und rechts vom Doppelpunkt sind gleich zwei vorhanden.

Nein, natürlich ist alles richtig: Der erste Punkt steht für den Code $2E, der zweite symbolisiert das nicht-druckbare Byte $0D und erscheint nicht wirklich. Hier kann man gut erkennen, daß in der ASCII-Darstellung Punkt nicht gleich Punkt ist; auf das Hex-Byte kommt es an (nur ein $2E-Byte ist ein echter Punkt).

Gezielte Suche nach Bytes über den H-Befehl

Wenn Sie ein Programm mit dem M-Befehl durchforsten und/oder ändern wollen, werden Sie bald merken, daß die Suche nach bestimmten Stellen innerhalb des Programms auf die Dauer recht mühselig ist.

H ist die Abkürzung des englischen Verbs »to hunt«, welches »jagen« heißt.

Die Syntax ist »H Anfangsadresse Endadresse Byte1 Byte2 Byte3 ... ByteX« (Kurzübersicht zum H-Befehl in Bild 8), wobei die Anzahl der Bytes frei wählbar ist; mindestens ein Byte muß im durch Anfangs- und Endadresse umgrenzten Speicherbereich gesucht werden. Die Obergrenze für die Anzahl der Bytes ist einfach die Zeilenlänge (zwei Zeilen auf dem 80-Zeichen-Bildschirm, bei 40-Zeichen vier).

Der Monitor gibt auf den H-Befehl hin alle die Adressen im fünfstelligen Hexadezimalformat aus, ab denen die gewünschte Bytefolge gefunden wurde. Sehen wir uns den neuen Befehl, auf den Sie bald nicht mehr verzichten wollen, an einem Beispiel an. Geben Sie dazu bitte »H F0000 F5000 52 45 53 54« ein.

Der Monitor gibt dann die Adressen F0A80 und F4443 als Ergebnis seiner Suche aus. Mit M können Sie nachprüfen, ob an den angegebenen Adressen wirklich die Bytes $52 $45 $53 $54 stehen. Zur Adresse F0A80 ist eine wichtige Anmerkung zu machen:

Wenn die Adresse x0A80 (x deshalb, weil die Speicherbank bei Adressen, die in dem Bereich liegen, der allen Bänken gemeinsam ist, uninteressant ist) nach dem H-Befehl auf dem Bildschirm erscheint, ignorieren Sie bitte diese Angabe! Sie werden feststellen, daß immer, wenn die Adressen ab $0A80 im Suchbereich liegen, x0A80 als Ergebnis gemeldet wird. Ab dieser Adresse merkt sich nämlich der Monitor die Bytes, die gesucht werden sollen. Klar, daß ein Vergleich dieses Monitor-Suchpuffers mit sich selbst ein positives Suchergebnis ergibt!

Eigentlich gehört dieser Hinweis ins Handbuch, aber auch die Anleitungen der uns bekannten C 64-Monitore enthalten keinen ähnlichen Hinweis. Die Adresse des »Suchpuffers« ist von Monitor von Monitor verschieden; dieser läßt sich jedoch ausfindig machen, und wir möchten Ihnen an dieser Stelle das Verfahren vorstellen, mit dem man den Bereich des Puffers feststellen kann. Dieses Verfahren wird für Sie eine gute Übung im Umgang mit dem Monitor-Kommando H sein.

Suchen Sie nach einer seltenen Bytefolge wie $D0 $E0 $F0 $00 $10 im ganzen Speicher:

H 00000 FFFFF D0 E0 F0 00 10

Nacheinander werden Sie alle möglichen Adressen erhalten, die sich (mit 99prozentiger Sicherheit) nur in der Speicherbank unterscheiden, die Adreßangabe wird (fast) immer gleich sein. Sollten nicht alle angezeigten Adreßangaben gleich sein, suchen Sie noch nach einer anderen Bytefolge wie $E1 $D1 $F1 $01 $11. Streichen Sie (am Bildschirm einfach mittels der DEL-Taste) alle Adreßangaben weg, die nicht bei beiden Suchbefehlen gemeldet wurden. Die einzige Adresse, die übrigbleibt, ist dann eindeutig die Anfangsadresse des Monitor-Suchpuffers.

Falls bei unserem Suchverfahren etwas unklar war, sei noch der Hinweis gegeben, daß wir die Angabe der Bank ignorieren müssen; $F0A80 ist also nichts anderes als $00A80 und $10A80.

Bei unserem ersten Beispiel dieses Abschnitts haben wir über »H F0000 F5000 52 45 53 54« übrigens nach der ASCII-Folge »REST« gesucht. Die Suche nach ASCII-Zeichen wird eine der häufigsten Anwendungen des H-Befehls sein. Damit Ihnen dies leichter fällt, ist im TEDMON auch die Mög-

Befehl:	H
Syntax:	H Anfangsadresse Endadresse Byte1 ... ByteX
oder	H Anfangsadresse Endadresse "Suchtext
Wirkung:	sucht nach den Bytes oder den ASCII-Codes des Suchtextes im Bereich von Anfangsadresse bis Endadresse.
englisches Wort:	Hunt (englisch jagen)

Bild 8. Kurzübersicht zum Befehl H

lichkeit gegeben, nach ASCII-Codes zu suchen. Diese wurde Ihnen bislang vorenthalten, aber jetzt sollen Sie es spätestens erfahren (in der Kurzübersicht wurde dafür diese Möglichkeit schon genannt).

Wenn Sie im Bereich von $0000 bis $5000 in Bank $F (dezimal 15) suchen lassen wollen, ist auch folgende Eingabe möglich:

H F0000 F5000 'REST

Das Apostroph signalisiert dem Monitor, daß ein ASCII-Text folgt, der vor der Suche in die entsprechenden Bytes umzuwandeln ist. Leider existiert diese Eingabemöglichkeit nur beim H-Befehl, obwohl man sie bei einigen anderen Kommandos sicher auch gerne hätte. Hier könnte man sich unter Umständen mit dem Suchpuffer behelfen, da die Zeichen in ihm schon in umgewandelter Form vorliegen.

Die ASCII-Codes

Schon beim M-Befehl wurden die ASCII-Codes angesprochen, allerdings ohne Erklärung.

Wie wir schon erwähnt haben, erkennt der Computer nicht die Zahlen und Zeichen so wie wir (wenn wir sie am Bildschirm sehen), sondern stellt Zahlen als elektronische Signale im Binärformat dar: Strom ein ist gleich »Bit gesetzt« (1), Strom aus entspricht »Bit gelöscht« (0). 8 Bit zusammengefaßt erlauben 256 verschiedene Kombinationen (1 bis 255 und die 0), und ein solches Byte kann mit zwei Hexadezimalziffern dargestellt werden. Da der Computer zunächst einmal nur mit Zahlen rechnen kann, stellt er auch Zeichen (wie die Buchstaben und Grafikzeichen, aber auch die Ziffern wie 0, 1, ... 9) als elektronische Signale – effektiv also als Zahlen – dar.

Dazu ein Beispiel:

A = %01000001 = $41 = #65 (# verwenden wir fortan als Symbol für Dezimalzahlen, da es das geläufigste ist. Dieses Symbol kennt der TEDMON jedoch nicht).

```
100 REM *********************
110 REM *                   *
120 REM *  ASCII - TABELLE  *
130 REM *                   *
140 REM *********************
150 REM *                   *
160 REM *  FLORIAN MUELLER  *
170 REM *                   *
180 REM *********************
190 :
200 DO : INPUT "DRUCKER (D) ODER BILDSCHIRM (B)"
    ;A$: LOOP WHILE A$<>"D" AND A$<>"B"
210 IF A$="D" THEN BEGIN
220 INPUT "GERAETENUMMER";G: INPUT "SEKUNDAERADR
    ESSE";S
230 OPEN 4,G,S: CMD 4
240 BEND
250 :
260 LE$=CHR$(32): REM LEERZEICHEN
270 Z$="0123456789ABCDEF": REM ZIFFERN FUER HEX-
    ZAHLEN
280 PRINT SPC(2);
290 :
300 FOR J = 1 TO 16
310 : PRINT MID$(Z$,J,1);LE$;
320 NEXT J
330 PRINT
340 :
350 FOR J = 0 TO 15
360 : PRINT MID$(Z$,J+1,1);SPC(5)
370 : FOR K = 2 TO 15: IF K>7 AND K<10 THEN 400
380 : : PRINT CHR$(J+(K*16));LE$;
390 : : IF K=7 THEN PRINT LE$;LE$;LE$;LE$;
400 : NEXT K
410 : PRINT
420 NEXT J
430 :
440 IF A$="D" THEN PRINT# 4: CLOSE 4
```

Listing 2. Erstellen einer ASCII-Tabelle

Statt PRINT"A" können wir in Basic auch PRINT CHR$(65) oder PRINT CHR$(DEC("41")) schreiben.

Der CHR$-Befehl gibt nämlich zu einem ASCII-Code das betreffende Zeichen aus. Wenn Sie unser Beispielprogramm »M-Simulation« (Listing 1) ansehen, finden Sie bei der Ausgabe der ASCII-Darstellung, die in den Zeilen 340 bis 380 erfolgt, den CHR$-Befehl wieder.

Das Gegenstück ist der ASC-Befehl: PRINT ASC("A") ergibt 65. ASC sind die ersten drei Buchstaben der Abkürzung ASCII. ASCII heißt »American Standard Code for Information Interchange«, übersetzt ins Deutsche »Amerikanische Standardcodierung für den Datenaustausch«. Das Ziel dieser Codierung war, daß verschiedene Geräte Zeichen untereinander ohne Probleme austauschen können. Leider ist dieses Vorhaben gescheitert, denn viele Hersteller weichen davon ab und verwenden eigene Codierungen. Wenigstens sind die ASCII-Codes der Commodore-Geräte untereinander einheitlich, sonst könnten Sie nicht einmal ein C 64-Basic-Programm im C 128-Modus laden.

Im Handbuch finden Sie im Anhang eine Tabelle der ASCII-Codes. Listing 2 gibt eine hexadezimale Tabelle der ASCII-Codes aus. Lassen Sie sich am besten die Tabelle auf dem Drucker ausgeben und hängen diese in der Nähe Ihres Computers auf. Sie werden diese Tabelle noch oft brauchen.

Steuerzeichen	ASCII-Code
Unterstreichen ein *	$02 = # 2
Zeichenfarbe auf weiß stellen	$05 = # 5
Klingelzeichen	$07 = # 7
Tabulator	$09 = # 9
Zeilenvorschub LINE FEED (LF)	$0A = # 10
SHIFT/C= blockieren	$0B = # 11
SHIFT/C= wieder entriegeln	$0C = # 12
CARRIAGE RETURN (Wagenrückl.)	$0D = # 13
Text-Zeichensatz ein	$0E = # 14
Blinkmodus ein *	$0F = # 15
CURSOR DOWN (Cursor runter)	$11 = # 17
REVERS ON (Revers ein)	$12 = # 18
HOME (Cursor nach links oben)	$13 = # 19
DELETE (Zeichen löschen)	$14 = # 20
TAB (Tabulator setzen/löschen)	$18 = # 24
ESCAPE (Sonderfunktionen)	$1B = # 27
CURSOR RIGHT (Cursor rechts)	$1D = # 29
Zeichenfarbe orange	$81 = #129
Unterstreichen aus *	$82 = #130
SHIFT RETURN	$8D = #141
Grafik-Zeichensatz ein	$8E = #142
Blinkmodus aus *	$8F = #143
Zeichenfarbe schwarz	$90 = #144
CURSOR UP (Cursor rauf)	$91 = #145
REVERS OFF (Revers aus)	$92 = #146
CLEAR (Bildschirm löschen)	$93 = #147
INSERT (1 Zeichen einfügen)	$94 = #148
Zeichenfarbe braun	$95 = #149
Zeichenfarbe hellrot	$96 = #150
Zeichenfarbe grau 1	$97 = #151
Zeichenfarbe grau 2	$98 = #152
Zeichenfarbe hellgrün	$99 = #153
Zeichenfarbe hellblau	$9A = #154
Zeichenfarbe grau 3	$9B = #155
Zeichenfarbe purpur	$9C = #156
CURSOR LEFT (Cursor links)	$9D = #157
Zeichenfarbe gelb	$9E = #158
Zeichenfarbe türkis	$9F = #159

Bild 9. Steuerzeichen und ihre ASCII-Codes

Diese Tabelle umfaßt allerdings nicht die sogenannten Steuerzeichen. Zu den Codes $00 bis $1F sowie $80 bis $9F sind darum auch keine Zeichen ausgegeben. Bei $20, $A0 und $E0 finden Sie auch kein Zeichen. Hier handelt es sich jedoch nicht um Steuerzeichen, sondern um die Codes für das Leerzeichen.

Steuerzeichen – auch nichtdruckbare Zeichen genannt – lassen sich auch über PRINT ausgeben. Sie erzeugen also nicht direkt ein Zeichen, sondern rufen eine Steuerfunktion hervor. PRINT CHR$(7) erzeugt beispielsweise das akustische Klingelzeichen.

Die Steuerfunktionen finden Sie in Bild 9 zusammengefaßt. Wenn wir mit dem Monitor Basic-Programme gegen LISTen schützen wollen, werden Sie merken, was sich mit den Steuerzeichen alles anstellen läßt!

Lediglich zum Steuerzeichen »ESC« möchten wir noch einige Hinweise geben. Wenn Sie den ESC-Code (#27 = $1B) senden, wird das nächste Zeichen, das über PRINT gesendet wird, als ESC-Anweisung (zum Beispiel »A« zum Einschalten des automatischen Einfügemodus) behandelt. Damit werden wir uns noch genauer beschäftigen.

Wir wollen aber zunächst nach ASCII-Zeichen suchen (mit dem H-Befehl). Dazu ist eine Umwandlung in den ASCII-Code erforderlich, die Ihnen mit einiger Übung leicht fallen wird.

Für viele Zeichen gibt es einfache Regeln, um den ASCII-Code zu bestimmen:

Bei Ziffern (0 bis 9) wird einfach $30 dazugezählt; Beispiele: 0 = $30, 5 = $35, 9 = $39.

Ähnlich geht es bei den Satzzeichen, die Sie über <SHIFT+Zifferntaste> erreichen (! = <SHIFT+1>):

In diesem Fall wird statt der $30 eine $20 dazugezählt. Beispiele: ! = <SHIFT+1> = $21; " = <SHIFT+2> = $22; (= <SHIFT+8> = $28.

Wenn wir nach dem Text »)1977« suchen wollen, der ja in der Einschaltmeldung des C 128 (»(C)1977 MICROSOFT CORP.«) vorkommt, geht dies folglich über H F4000 FF000 28 31 39 37 37.

Wir suchen nur im ROM-Bereich (Bank 15 ab $4000), da die Einschaltmeldung logischerweise dort steht (sie muß ja auch nach dem Ausschalten des Computers vorhanden sein und darf deshalb nicht im RAM stehen).

Die Eingabemodi des Betriebssystems

In diesem Abschnitt befassen wir uns ein wenig mit der Eingabe der Zeichen und der Umwandlung der Tastatureingaben in den ASCII-Code. Dies ist ein Thema, das zwar nur am Rande zum Bereich Monitor gehört, aber trotzdem viele interessante Informationen vermittelt.

Wenn Sie sich im Basic-Editor befinden, wird einfach jedes Zeichen, das Sie über Tastatur eingeben, direkt auf dem Bildschirm ausgegeben. Nebenbei wird noch der Cursor angezeigt.

Beispiel: Sie drücken auf die Taste <A>, und A erscheint auf dem Bildschirm. Sie drücken <SHIFT> und die <CLR/HOME>-Taste und der Bildschirm wird wie mit PRINT CHR$(147) gelöscht. Sie sehen also, daß hier jedes Zeichen – auch Steuerzeichen für Cursor etc. – ausgegeben wird. Eine Eingabe wird mit <RETURN> abgeschlossen.

Diesen Eingabemodus nennt man »normal mode« (normaler Modus), da – wie gesagt – jedes Zeichen direkt ausgegeben wird und Steuerzeichen wie CLR/HOME direkt ausgeführt werden.

Ein anderer Modus ist der sogenannte »quote mode« (Anführungszeichen-Modus oder Hochkomma-Modus). In diesem befindet man sich, wenn ein Anführungszeichen ausgegeben wird. Wird allerdings ein zweites ausgegeben, befindet man sich wieder im »normal mode«.

Befehl:	T
Syntax:	T Anfangsadresse Endadresse Zieladresse
Wirkung:	verschiebt Bereich von Anfangsadresse bis Endadresse zur Zieladresse.
englisches Wort:	Transfer (übertragen, verschieben)

Bild 10. Kurzübersicht zum Befehl T

Wie bei allen Eingabemodi wird die Eingabe mit <RETURN> abgeschlossen. Tippen Sie bitte einmal folgende Zeichen ein (im Basic-Editor, nicht im Monitor):

```
10 PRINT "
```

Drücken Sie nicht die <RETURN>-Taste! Denn wenn Sie sie drücken, so haben Sie Ihre Eingabe abgeschlossen. Es ist egal, in welchem Eingabemodus Sie die Zeichen eingegeben haben. »10 PRINT« wurde im »normal mode« eingegeben und nach dem »"« befanden Sie sich im »quote mode«.

Der große Unterschied zwischen »quote mode« und »normal mode« ist der, daß im »quote mode« Steuerzeichen (wie CLR/HOME) – mit Ausnahme von ESC und DEL – nicht ausgeführt werden, sondern als inverse Zeichen – meist Grafikzeichen – erscheinen. Statt des Bildschirmlöschens wird, wenn Sie nach der oben stehenden Eingabe auf <SHIFT+CLR/HOME> drücken, ein inverses Herz ausgegeben. Der Sinn des »quote mode« ist der, in Strings – die ja zwischen zwei Anführungszeichen stehen – Steuerzeichen einzubauen. Wenn Sie die Zeile 10 PRINT "inverses Herz" nämlich ausführen lassen, befindet sich das Programm nicht im »quote mode« und der Bildschirm wird gelöscht. Beim LISTen hingegen steht das Steuerzeichen in Anführungszeichen und wird daher invers dargestellt und nicht ausgeführt.

Ein wichtiger Hinweis noch zum »quote mode«: Er ist immer dann aktiv, wenn eine ungerade Zahl von Anführungszeichen ausgegeben wurde. Dabei ist es unbedeutend, ob diese in der Zwischenzeit mittels DEL gelöscht wurden.

Der dritte Eingabemodus ist der »insert mode«, der Einfügemodus. In diesen gelangen Sie durch Drücken von <SHIFT+INS/DEL>. Der Einfügemodus wird dann beendet, wenn pro gedrücktes <SHIFT+INS/DEL> ein Zeichen eingegeben – und damit eingefügt – wurde.

Innerhalb des Einfügemodus gelten die gleichen Regeln wie im Anführungsmodus. Wenn Sie im Normalmodus auf eine Steuertaste (Cursor-Tasten, CLR/HOME,..) drücken, wird die Steuerfunktion ausgeführt, im »quote mode« und im »insert mode« hingegen nicht – es erscheint zum Beispiel ein inverses Herz.

Im »insert mode« ist zusätzlich DEL außer Kraft gesetzt.

Der vierte und letzte Eingabemodus ist der »auto insert mode«, der automatische Einfügemodus, in den Sie über <ESC> und <A> gelangen und aus dem Sie mittels <ESC> und <C> wieder herauskommen. In diesem Modus sind die Steuertasten nach wie vor wirkungsvoll; es handelt sich also um einen Normalmodus, bei dem die Zeichen eingefügt werden. Dieser Eingabemodus ist für unsere Zwecke nicht so ergiebig und soll nicht weiter behandelt werden.

Von jetzt an werden wir neue Befehle kennenlernen, denn je mehr Werkzeug in Form von Monitorbefehlen zur Verfügung stehen, desto besser können wir als direkte Folgeerscheinung mit unserem C 128 arbeiten!

Der nächste Befehl dient zum Verschieben von Speicherbereichen. Dies ist eine der Grundfunktionen eines Monitors.

Man kann einen Monitor auch als Textverarbeitung für den Speicher, also als Speicherverarbeitung ansehen, die eine komfortable Behandlung des Speichers ermöglicht (bequemer als über POKE, PEEK und BANK). So wie eine Textverarbeitung eine Kopierfunktion für Textbereiche haben sollte, kommt man auch mal in die Lage, einen Speicherbereich kopieren zu müssen.

Der entsprechende Befehl des TEDMON heißt »T«. T kommt vom englischen Wort »transfer«, welches die deutsche Bedeutung »übertragen« hat. Die Syntax lautet:
T Anfangsadresse Endadresse Zieladresse

Anfangsadresse und Endadresse begrenzen den Originalbereich, der kopiert werden soll. Der Zielbereich wird nur durch die Zieladresse bestimmt, die die Anfangsadresse des Zielbereichs ist. Die Endadresse des Zielbereichs kann der Monitor selbst ausrechnen, denn die Länge des zu verschiebenden Bereichs ist ihm durch Anfangsadresse und Endadresse indirekt mitgeteilt worden.

Alle nötigen Informationen zum T-Befehl finden Sie in Bild 10 (Kurzübersicht).

Zum T-Befehl ist aber noch eines zu sagen: Maschinenprogramme sind nach dem Verschieben nicht unbedingt lauffähig! Die Programme würden nur dann laufen, wenn keine

```
100 REM ******************************
110 REM *                            *
120 REM * SIMULATION DES MONITORBE-  *
130 REM *                            *
140 REM *  FEHLS 'T' MITTELS BASIC   *
150 REM *                            *
160 REM ******************************
170 REM *                            *
180 REM *  1986 BY FLORIAN MUELLER   *
190 REM *                            *
200 REM ******************************
210 :
220 PRINT CHR$(13) CHR$(13)"VERSCHIEBEN MITTELS
    BASIC" CHR$(7) CHR$(13)
230 INPUT "SPEICHERBANK DES QUELLBEREICHS";BQ
240 INPUT "ANFANGSADRESSE D.QUELLBEREICHS";AQ$:
    AQ=DEC(AQ$)
250 INPUT "ENDADRESSE{3SPACE}DES QUELLBEREICHS";
    EQ$: EQ=DEC(EQ$)
260 INPUT "SPEICHERBANK DES{2SPACE}ZIELBEREICHS"
    ;BZ
270 INPUT "ANFANGSADRESSE D. ZIELBEREICHS";AZ$:
    AZ=DEC(AZ$)
280 EZ=AZ+(EQ-AQ): REM ENDE DES ZIELBEREICHS BER
    ECHNEN
290 PRINT "=> ENDADRESSE DES ZIELBEREICHS = $";H
    EX$(EZ)
300 :
310 IF AQ>AZ THEN 440: REM GEGEBENENFALLS ZWEITE
     SCHLEIFE ANSPRINGEN
320 :
330 Q=EQ: Z=EZ: REM Q = QUELLADRESSE; Z = ZIELAD
    RESSE
340 :
350 DO UNTIL Q<AQ
360 : BANK BQ: QUELL=PEEK(Q): REM QUELLBEREICH A
    USLESEN
370 : BANK BZ: POKE Z, QUELL: REM ZIELBEREICH BE
    SCHREIBEN
380 : IF PEEK(Z)<>QUELL THEN PRINT RIGHT$(HEX$(B
    Q),1);HEX$(Q),
390 : Q=Q-1: Z=Z-1: REM QUELL- UND ZIELADRESSE E
    RNIEDRIGEN
400 LOOP
410 :
420 RUN
430 :
440 Q=AQ: Z=EQ: REM Q=QUELL-,Z=ZIELADRESSE
450 :
460 DO UNTIL Q>EQ
470 : BANK BQ: QUELL=PEEK(Q): REM QUELLBEREICH A
    USLESEN
480 : BANK BZ: POKE Z, QUELL: REM ZIELBEREICH BE
    SCHREIBEN
490 : IF PEEK(Z)<>QUELL THEN PRINT RIGHT$(HEX$(B
    Q),1);HEX$(Q),
500 : Q=Q+1: Z=Z+1: REM QUELL- UND ZIELADRESSE E
    RHOEHEN
510 LOOP
520 :
530 RUN
```

Listing 3. T-Befehl in Basic simulieren

absoluten Adressierungen vorkommen; dies ist jedoch selbst bei Kleinstroutinen nur mit Mühe zu erreichen.

Leider kennt der TEDMON des C128 nicht die entsprechenden Verschiebebefehle des SMON zum C64, der das Anpassen eines Maschinenprogramms an einen neuen Speicherbereich in professioneller Manier löst.

Listing 3 ist ein Basic-Programm – darum ist es so langsam – zur Simulation des T-Befehls mittels Basic.

Sie können im Programm die Formel zur Berechnung der Endadresse des Zielbereichs gut erkennen.

Die Länge von Listing 3 zeigt, daß die Verschiebefunktion nicht ganz so einfach zu programmieren ist. Vom Verhältnis Quellbereich/Zielbereich bezüglich der Lage im Speicher hängt nämlich die Form der Schleifenkonstruktion ab.

Im übrigen kann es passieren, daß bei T-Befehlen wie »T 00000 01000 FA000« der Monitor (beziehungsweise das Beispielprogramm) bestimmte Adressen ausgibt. Diese Adressen sind Adressen, die beim Verschieben Schwierigkeiten gemacht haben, weil der zu den Adressen gehörende Zielbereich nicht als RAM verwendbar ist.

Die Adressen werden ausgegeben, wenn man versucht, ROM-Bereiche als Zielbereiche zu verwenden (im Beispiel »T 00000 01000 FA000 wäre der Bereich ab $A000 in Bank $F (dezimal 15) als Zielbereich gewählt worden, obwohl es sich um einen ROM-Bereich handelt). Dieser Fehler wird dadurch gefunden, daß der Monitor beziehungsweise das Beispielprogramm nach jedem Byte prüft, ob der Zielbereich an der Stelle des gerade verschobenen Bytes auch mit dem Originalbereich übereinstimmt. Bei RAM-Bereichen, die man ja auch beschreiben und somit als Zielbereich einsetzen kann, tritt natürlich eine Übereinstimmung auf; bei ROM-Bereichen, die als Festwertspeicher nur les-, nicht aber veränderbar sind, führt ein solches Überprüfen (Verify) zu einem Fehler. Gleiches gilt, wenn man beim M-Befehl – oder mittels des Unterbefehls > – einen ROM-Bereich zu ändern versucht.

Bei Eingabefehlern erweist es sich als sehr nützlich, daß der Monitor die Adressen meldet; man muß aber bedenken, daß beim Transport in ROM-Bereiche aufgrund der Prozessor- und Speicherverwaltungsstruktur nicht das ROM, sondern das an gleicher Adresse liegende RAM beschrieben wird. So könnten Programme oder Programmteile verlorengehen, ohne daß man es beabsichtigt hat. Wenn man aber anhand der ausgegebenen Adressen merkt, daß ein Fehler zustandekam, kann man noch schnell mit <RUN/STOP> abbrechen und somit den Schaden begrenzen.

Noch etwas zum Verifizieren:

Leider ist das Verifizieren – ob beim Monitor oder beim Simulationsprogramm – nicht 100prozentig sicher: Wenn Sie den ROM-Bereich $4000 bis $A000 (in Bank 15) sich selbst kopieren lassen (erforderliches Monitor-Kommando: T F4000 FA000 F4000), könnte der Monitor den Fehler nie ausfindig machen; er würde immer den Bereich $4000 bis $A000 mit dem Bereich selbst vergleichen und logischerweise keinen Fehler entdecken. Der Computer schreibt aber den ROM-Bereich dann nicht ins ROM selbst (das geht ja nicht), sondern ins darunterliegende RAM. Das heißt, effektiv würde der Bereich $F4000 bis $FA000 in den Bereich $04000 bis $0A000 kopiert. Prüfen Sie das doch einmal nach, indem Sie mittels M (den Vergleichsbefehl des Monitors haben wir noch nicht durchgenommen) den Bereich $4000 bis $A000 in Bank 0 mit dem entsprechenden Bereich in Bank 15 ($F) vergleichen, nachdem Sie über »T F4000 FA000 F4000« das Verschieben durchgeführt haben.

Beim Kopieren mit »T F4000 FA000 F4001« würde der Monitor hingegen Fehler entdecken, da der Zielbereich ($F4001 bis $FA001) nun ein anderer Speicherbereich wäre als der Originalbereich ($F4000 bis $FA000).

Noch ein Hinweis, um Mißverständnissen vorzubeugen: Selbstverständlich kann man ROM-Bereiche ins RAM an anderer Stelle kopieren, nur nicht umgekehrt. So kann man den Bereich $F4000 bis $FA000 zwar nicht nach $F4001 bis $FA001 kopieren, wohl aber nach $04001 bis $0A001 verschieben: T F4001 FA001 04001.

Der T-Befehl allein ist zunächst einmal nicht so recht verwertbar, vor allem erscheint einem der Nutzen des Kommandos dann zweifelhaft, wenn man nicht Maschinenprogramme bearbeiten will.

Wir möchten Ihnen zur Entschädigung ein Verfahren vorstellen, um die Lage des Basic-Programms im Speicher zu ändern, also kurz gesagt: Das Verschieben des Basic-Programms in einen anderen Bereich.

Bitte lösen Sie dazu einen Reset aus und laden dann ein Basic-Programm Ihrer Wahl in den Speicher. Es sollte der Einfachheit halber nicht allzulang sein.

Wir verwenden als Beispielprogramm das Programm »CHANGE UNIT« von der Test/Demo-Diskette zur 1570/1571 und laden es über DLOAD"CHANGE UNIT" <RETURN>. Das Programm soll nun nach $A000 (in Bank 0) verschoben werden.

Am besten arbeiten wir nun gemeinsam an diesem Beispielprogramm, das wohl jeder C128-Besitzer mit einer der neuen Commodore-Floppy-Stationen zur Verfügung hat, das Verschieben dieses Programms durch. Daran können Sie natürlich auch lernen, wie man beliebige Programme verschiebt. Die einzelnen Schritte unseres Vorgehens numerieren wir mit römischen Zahlen.

I. Bestimmung von Anfangs- und Endadresse

Geben Sie dazu folgenden Audruck ein (? ist die Abkürzung für PRINT):

```
?HEX$(PEEK(45)+256*PEEK(46)),HEX$(PEEK(4624)+256*P-
EEK(4625))
```

Sie erhalten die hexadezimale Darstellung von Anfangs- (linker Wert) und Endadresse (rechter Wert). Hexadezimale Darstellung deshalb, weil wir mit ihr an einigen Stellen besser zurechtkommen.

Bei unserem Beispielprogramm CHANGE UNIT von der Test/Demo-Diskette erhalten wir den folgenden Ausdruck:
1C01 20F5

Es wird nicht erwähnt, daß es sich um Adressen in Bank 0 handelt, da dies von Anfang an feststeht: Bank 0 ist die Speicherbank, in der das Basic-Programm abgelegt ist.

Die Anfangsadresse ist in der Regel $1C01 (wie bei unserem Beispiel) oder $4001 bei Benutzung der Befehle zur hochauflösenden Grafik, die Endadresse – im Beispiel $20F5 – variiert natürlich (Programme sind ja auch unterschiedlich lang).

Diese Werte müssen wir uns natürlich merken, denn wenn wir das Basic-Programm verschieben wollen, müssen wir dem Monitor Anfangs- und Endadresse des Basic-Programms im Speicher mitteilen (unser Monitor ist schließlich kein Hellseher, wir haben ihn mit einer ausreichenden Anzahl von Informationen zu versorgen).

II. Bestimmung der Endadresse des Zielbereichs

Vielleicht stutzen Sie zunächst ein wenig, warum denn die Endadresse des Zielbereichs berechnet werden soll, wenn dies der Monitor ohnehin tut.

Die Bestimmung ist auch nur erforderlich, damit wir die Endadresse in bestimmten Adressen, die der Basic-Interpreter benötigt, ablegen können. Die Endadresse des Zielbereichs ergibt sich über die Formel
Endadresse Zielbereich = Endadresse – Anfangsadresse + Zieladresse.

Wir errechnen den hexadezimalen Wert über den Ausdruck ?HEX$(DEC("hier Hex-Endadresse aus Schritt I einsetzen") - DEC("hier Hex-Anfangsadresse aus Schritt I einsetzen") + DEC("hier Hex-Endadresse des Zielbereichs einsetzen"))

In unserem konkreten Fall »CHANGE UNIT nach $A000« heißt dies folgendermaßen:

```
?HEX$(DEC("20F5")-DEC("1C01")+DEC("A000"))
```

und bringt folgendes Ergebnis auf den Bildschirm:
A4F4

Nun haben wir nach den Schritten I und II alle wichtigen Parameter errechnet:

Anfangsadresse (Originalbereich) = $1C01
Endadresse (Originalbereich) = $20F5
Anfangsadresse (Zielbereich) = $A000
Endadresse (Zielbereich) = $A4F4

III. Zeiger des Interpreters setzen

Nun müssen wir die Zeiger des Interpreters auf Anfangs- und Endadresse des Basic-Programms im Speicher an die Werte nach dem Verschieben (das Verschieben ist dann der nächste Schritt) anpassen. Außerdem ist ein Null-Byte als Markierung vor den neuen Basic-Start (das Programm soll nun bei $A000 liegen) anzubringen.

Vorher noch etwas zur Wiederholung des Hexadezimalsystems:

Das Low-Byte einer vierstelligen Hexadezimalzahl besteht aus den beiden am weitesten rechts stehenden Ziffern, die beiden am weitesten links stehenden Ziffern kennzeichnen das High-Byte. Beispiel: Zahl = $ABCD; LB (Abkürzung für Low-Byte) = $CD, HB (Abkürzung für High-Byte) = $AB.

Da in Adressen (eine Speicherzelle kann nur ein Byte aufnehmen) das Low-High-Format verwendet wird (eine Adresse – die der Adreßangabe nach niedrigere – enthält das Low-Byte, die andere – die der Adreßangabe nach höhere – enthält das High-Byte), sind in unserer nächsten Anweisungszeile folgende Rechnungen erforderlich:

POKE 45, DEC("LB der neuen Anfangsadresse"): POKE 46, DEC("HB der neuen Anfangsadresse"): POKE DEC("neue Anfangsadresse")-1, 0: POKE 4624, DEC("LB der neuen Endadresse nach Schritt II"): POKE 4625, DEC("HB der neuen Endadresse gemäß II"

Bei unserem Beispiel lautet die Eingabe folgendermaßen

```
POKE45,DEC("00"):POKE46,DEC("A0"):POKE DEC("A000")-1,0:POKE4624,DEC("F4"):POKE4625,DEC("A4")
```

Nun weiß der Basic-Interpreter, in welchem Bereich das Basic-Programm liegt.

IV. Verschieben des Programms

Während wir bislang im Basic hantiert haben, kommt nun der Monitor zu seinem großen Auftritt. Wir verschieben das Basic-Programm an die Zieladresse. Dazu sind die in I. ermittelten Parameter (Anfangs- und Endadresse des Originalbereichs) sehr nützlich. Vorher aber schalten wir den Monitor mittels <F8> oder Basic-Befehl MONITOR ein.

Die Anweisung an den Monitor lautet:
T Anfangsadresse Endadresse Zieladresse

Im Beispiel »CHANGE UNIT nach $A000« heißt das also:
T 1C01 20F5 A000

Die Bank-Angabe lassen wir einfach weg, wir arbeiten ja in diesem Fall ohnehin nur in Bank 0 – und diese Bank ist voreingestellt.

Nach dem Verschieben verlassen wir den Monitor wieder mit X.

V. Basic-Programm an neuen Bereich anpassen

Wenn Sie nun versuchen, das Programm zu LISTen, werden Sie merken, daß es an seinen neuen Speicherbereich noch nicht gänzlich angepaßt ist. Dies ist jedoch im Gegensatz zum Anpassen von Maschinenprogrammen nicht weiter schwierig. Wir greifen dazu auf eine ROM-Routine zurück, die alle notwendigen Änderungen auf Wunsch durchführt:

```
BANK15:SYS20303
```

Diese Routine wird übrigens nach dem Laden eines Programms von der Kassette oder der Diskette immer vollautomatisch ausgeführt.

Jetzt haben wir es geschafft! In Zukunft werden Sie sich bei diesem Anwendungsfall viel leichter tun. Sie brauchen nur hier nachzuschlagen. Im übrigen geht es bei einem Basic-Programm, das Sie auf Diskette gespeichert haben, viel einfacher.

Der neue Beginn des aktuellen Basic-Programms im Speicher wird folgendermaßen festgesetzt:

POKE 45, DEC("LB neuer Start"): POKE 46, DEC("HB neuer Start"): POKE DEC("Hex-Adresse für neuen Start")-1, 0: NEW

Jetzt laden Sie das Programm mit DLOAD "NAME".

Natürlich kann das Programm auch auf einem anderen Speichermedium als der Diskette abgelegt sein.

Das vorgestellte erste Verfahren (unter Verwendung des T-Befehls) ist hauptsächlich interessant, wenn das Programm nicht auf Diskette vorhanden ist. Dies kann zum Beispiel bei Zwischenversionen der Fall sein.

Vor allem aber sollten Sie sehen, daß man den T-Befehl nicht nur für Maschinenprogramme, sondern auch für Basic nutzen kann; man muß nur wissen, wie das geht – dieser Kurs versucht, Ihnen alle nötigen Informationen zu vermitteln.

Der Vergleichs-Befehl: C

Im vorangegangenen Abschnitt wurde schon ein Vergleichs-Befehl erwähnt. In diesem Abschnitt sollen Sie alle nötigen Informationen zum Vergleichs-Befehl erhalten.

Der Vergleichs-Befehl lautet C (vom englischen Wort »compare«, deutsche Bedeutung »vergleichen«) und stimmt in der Syntax mit dem T-Befehl überein:
C Anfangsadresse Endadresse Zieladresse

Anfangsadresse und Endadresse markieren den ersten Bereich, Zieladresse ist die Anfangsadresse des zweiten Bereichs. Dann werden die verschiedenen Bereiche miteinander verglichen; unterschiedliche Adressen werden hexadezimal dargestellt.

Natürlich ergibt sich auch hier mit Hilfe einer Gleichung die Endadresse des Zielbereichs:
Endadresse des Zielbereichs = Endadresse – Anfangsadresse + Zieladresse.

Diese Gleichung werden Sie in Listing 4 – unserem Simulationsprogramm zum Monitor-Kommando – in Zeile 280 wiederfinden.

In Bild 11 ist die Kurzübersicht zum C-Befehl – auch diese ist Ihnen sicher zur Gewohnheit geworden – dargestellt.

Zur Berechnung der Endadresse des zweiten Bereichs ist noch etwas zu sagen: es ist nicht so, daß sie unbedingt vom Monitor berechnet wird. Er zählt einfach – wie das Beispielprogramm – regelmäßig die erste und die zweite Adresse um eins herauf. Die Vergleiche führt er mit der Adresse durch, deren Endwert ihm durch die Parameterübergabe bekannt ist (also der ersten Adresse). Beim C-Befehl wird also nur immer die Adresse des ersten Bereichs mit dem Endwert des ersten Bereichs verglichen. Eine zusätzliche Prüfung der Adresse des zweiten Bereichs erübrigt sich.

Wenn Sie einen Bereich mit sich selbst vergleichen, wird natürlich keine unterschiedliche Adresse gemeldet (Beispiel: C F4000 F5000 F4000).

Der C-Befehl ist vor allem für Maschinenprogrammierer interessant, die verschiedene Versionen ihrer Programme durch den C-Befehl miteinander vergleichen lassen und somit Unterschiede zwischen den einzelnen Versionen leicht entdecken.

```
100 REM *****************************
110 REM *                           *
120 REM * SIMULATION DES MONITORBE- *
130 REM *                           *
140 REM *  FEHLS 'C' MITTELS BASIC  *
150 REM *                           *
160 REM *****************************
170 REM *                           *
180 REM *  1986 BY FLORIAN MUELLER  *
190 REM *                           *
200 REM *****************************
210 :
220 PRINTCHR$(13)CHR$(13)"VERGLEICHEN MITTELS B
ASIC"CHR$(7)CHR$(13)
230 INPUT "SPEICHERBANK DES 1. BEREICHS";B1
240 INPUT "ANFANGSADRESSE D.1. BEREICHS";A1$:A1
=DEC(A1$)
250 INPUT "ENDADRESSE   DES 1. BEREICHS";E1$:E1
=DEC(E1$)
260 INPUT "SPEICHERBANK DES 2. BEREICHS";B2
270 INPUT "ANFANGSADRESSE D.2. BEREICHS";A2$:A2
=DEC(A2$)
280 E2=A2+(E1-A1):REM ENDE DES ZIELBEREICHS BER
ECHNEN
290 PRINT"=> ENDADRESSE DES 2. BEREICHS = $";HE
X$(E2)
295 PRINT:PRINT"NICHT UEBEREINSTIMMENDE ADRESSE
N:"CHR$(13)
300 :
310 T1=A1:T2=A2:REM ZU TESTENDE ADRESSEN BESTIM
MEN (MIT ANFANGSWERTEN BEGINNEN)
320 :
330 DO UNTIL T1>E1
340 :BANK B1:W1=PEEK(T1):REM WERT AUS BEREICH 1
 AUSLESEN
350 :BANK B2:W2=PEEK(T2):REM WERT AUS BEREICH 2
 AUSLESEN
360 :IF W1<>W2 THEN PRINT RIGHT$(HEX$(B1),1);HE
X$(T1),:REM KEINE UEBEREINSTIMMUNG, DANN ADRESS
E AUSGEBEN
370 :T1=T1+1:T2=T2+1:REM ADRESSEN ERHOEHEN
380 LOOP
390 :
400 RUN
```

Listing 4. Auch der C-Befehl kann in Basic simuliert werden

Befehl:	C
Syntax:	C Anfang 1 Ende1 Anfang2
Wirkung:	vergleicht den Bereich von Anfang1 bis Ende1 mit dem Bereich ab Anfang2.
englisches Wort:	Compare (vergleichen)

Bild 11. Kurzübersicht zum Befehl C

Der Füll-Befehl: F

Die Befehlsabkürzung F kommt vom englischen Wort »fill« (deutsche Bedeutung »füllen«).

Die Syntax: F Anfangsadresse Endadresse Füll-Byte

Im Gegensatz zur Anfangs- und Endadresse darf das Füll-Byte (das Byte, mit dem der durch Anfangsadresse und Endadresse markierte Bereich aufgefüllt werden soll) nur einen Byte-Wert ($00 bis $FF = #0 bis #255 = %00000000 bis %11111111) haben.

Der Befehl bewirkt, daß alle Speicherzellen von Anfangsadresse bis einschließlich Endadresse den Wert des Füll-Bytes annehmen. Am besten läßt sich dies an Listing 5, dem Simulationsprogramm zum F-Befehl, erkennen.

Dieses Programm erkennt übrigens einen Fehler, wenn man einen ROM-Bereich füllen will, und steigt mit der Meldung »??? fehlerhafte Adresse« aus. Der Monitor-Befehl F tut dies nicht. Um das Beispielprogramm also kompatibel zu gestalten, ist die Zeile 320 einfach zu löschen.

An der Kürze von Listing 5 kann man sehen, wie einfach und elementar zugleich der F-Befehl ist.

Bild 12 ist die Kurzübersicht zum F-Befehl.

Auch die Anwendung des Fill-Befehls ergibt sich aus der jeweiligen Situation heraus. Der Fill-Befehl kann zum Beispiel verwendet werden, um eine ASCII-Tabelle zu löschen. Dazu überschreibt man sie mit Leerzeichen (Füll-Byte $20 = #32).

So kann man den 40-Zeichen-Bildschirmspeicher mit folgender Eingabe löschen (vor der Eingabe bitte in die 40-Zeichen-Darstellung gehen):

F+1024+2023+32

Mit folgendem Befehl wird er durch Sterne überschrieben:

F+1024+2023+42

Dieses Beispiel dient eigentlich mehr der Demonstration als der wirklichen Anwendung. Wenn Sie sich jetzt im 40-Zeichen-Modus befinden und einen Befehl an den Monitor senden wollen, sollten Sie entweder die Eingabezeile hinter dem Monitor-Befehl über <ESC> und <Q> löschen oder einen Doppelpunkt »:« hinter den Befehl setzen. Schon beim Ändern des Memory-Dumps wurde erwähnt, daß dieser Doppelpunkt die Endmarkierung eines Monitor-Befehls ist – hier haben Sie eine denkbare Anwendung.

Eine häufigere Anwendung ist aber das Löschen des Basic-Programm-RAMs in Bank 0. Am besten überschreibt man es dazu mit Null-Bytes:

F 1C00 FDFF 00

Mit »H 1C00 FDFF 00« erhalten Sie dann alle Adressen von 1C00 bis FDFF, da ja der gesamte Suchbereich nur aus Null-Bytes (dem ehemaligen Füll- und jetzigen Such-Byte) besteht. So wie man mit dem C-Befehl die Arbeit des T-Befehls prüfen kann, ist auch eine – sich eigentlich erübrigende – Prüfung des F-Befehls mittels H möglich. Diese Prüf-


```
100 REM *****************************
110 REM *                           *
120 REM * SIMULATION DES MONITORBE- *
130 REM *                           *
140 REM *  FEHLS 'F' MITTELS BASIC  *
150 REM *                           *
160 REM *****************************
170 REM *                           *
180 REM *  1986 BY FLORIAN MUELLER  *
190 REM *                           *
200 REM *****************************
210 :
220 PRINTCHR$(13)CHR$(13)"FUELLEN MITTELS BASIC
"CHR$(7)CHR$(13)
230 INPUT "SPEICHERBANK DES FUELLBEREICHS";B
240 INPUT "ANFANGSADRESSE D.FUELLBEREICHS";A$:A
=DEC(A$)
250 INPUT "ENDADRESSE   DES FUELLBEREICHS";E$:E
=DEC(E$)
260 INPUT "FUELLBYTE";F$:F=DEC(F$)
270 :
280 FA=A:REM FUELLADRESSE INITIALISIEREN
290 :
300 DO UNTIL FA>E
310 :BANK B:POKE FA,F
320 :IF PEEK(FA)<>F THEN PRINT"??? ";RIGHT$(HEX
$(B),1);HEX$(FA):EXIT
330 :FA=FA+1
340 LOOP
350 :
360 RUN
```

Listing 5. F-Befehl als Simulation in Basic

Befehl:	F
Syntax:	F Anfangsadresse Endadresse Füll-Byte
Wirkung:	füllt Speicherbereich von Anfangsadresse bis Endadresse mit dem Füll-Byte auf.
englisches Wort:	Fill (füllen)

Bild 12. Kurzübersicht zum Befehl F

verfahren unterscheiden sich jedoch in einem grundsätzlichen Punkt: Während der T-Befehl nur korrekt gearbeitet hat, wenn der C-Befehl mit gleichen Parametern keine Adresse auf den Bildschirm bringt, ist die Arbeit des F-Befehls nur in Ordnung, wenn ein H-Befehl mit gleichen Parametern alle Adressen im Suchbereich ausgibt.

Im übrigen dient das Prüfen der Monitor-Befehle nicht dem Test der ohnehin korrekten Monitor-Routinen, sondern vielmehr der Suche nach Änderungen, die man nach bestimmten Befehlen durchgeführt hat.

Ein wichtiger Hinweis zum Beispiel »F 1C00 FDFF 00«: Im Gegensatz zum NEW-Befehl oder Reset ist nach dieser Eingabe ein Retten des Basic-Programms durch ein OLD-Programm (64'er, Ausgabe 12/85, Seite 43) nicht möglich, da der Speicher wirklich gelöscht wird. Beim NEW-Befehl oder Reset hingegen bleibt der Großteil des Speichers erhalten, da nur 3 Byte gelöscht werden.

Noch eine Anwendung, die sicher auch auftritt, wenn Sie den C64-Modus ab und zu nutzen:

Wenn ein GO64 zum Absturz führt, da der C64-Modus gegen Reset geschützt ist, kann man

a) die Radikalkur anwenden, indem man den gesamten Speicher in Bank 0 wie beschrieben löscht und dann GO64 eingibt oder
b) mit »F 8004 8008« die Markierung löschen, die für den Reset-Schutz verantwortlich ist, und dann »GO64« eingeben.

Das Verfahren b) ist natürlich schneller (es müssen nur wenige Bytes überschrieben werden). Das Verfahren a) ist oft aber empfehlenswert, wenn sichergestellt werden soll, daß sich nicht noch ein altes Programm im Speicher befindet.

Der Disketten-Operationsbefehl: @

Der Befehlsbuchstabe für Diskettenoperationen ist der Klammeraffe. Der Klammeraffe ist das Zeichen, das Sie auf der amerikanischen Tastatur rechts vom P finden. Es handelt sich um ein A in einer Art Kreis, der es umklammert.

Die Umsteiger vom C64 auf den C128 kennen den Klammeraffen als Disketten-Operationsbefehl schon vom DOS 5.1. Seine Syntax ist in einigen Punkten auch beim TEDMON vorhanden.

Die Syntax des @-Befehls ist etwas schwierig zu beschreiben, da auf diesen Befehl mehrere Funktionen fallen.

Der Befehl kann nur aus dem »Klammeraffen« bestehen, dann wird der Status von Laufwerk 8 angezeigt. Um Gerät 9 anzuwählen, gibt es die Möglichkeit, direkt nach dem Klammeraffen die Gerätenummer anzugeben: »@9« ist der gleiche Befehl, spricht aber Laufwerk 9 an.

Wenn Sie nach dem Befehl und der eventuellen Angabe der Gerätenummer, die selbstverständlich entfallen kann, ein Komma »,« setzen und darauf ein Disk-Befehl folgt, wird dieser ausgeführt.

Beispiel: »@,I« initialisiert Laufwerk 8 (da zwischen dem Klammeraffen und dem Komma keine Zahl steht), »@9,I« ist das gleiche für Laufwerk 9.

Übrigens: Weder der Monitor-Befehl noch einzelne Parameter nach dem Klammeraffen werden in Anführungszeichen gesetzt!

Beginnt der gegebene Befehl nach dem Klammeraffen mit dem Dollarzeichen »$«, so handelt es sich natürlich nicht um einen Disk-Befehl. TEDMON interpretiert dies als Directory-Befehl: »@8,$« listet das Directory von Laufwerk 8. Dem »$« können auch weitere Zeichen folgen, so daß ein Selektieren bei der Ausgabe des Directories kein Problem ist:

»@,$:*=PRG« gibt nur die Programm(PRG)-Files aus.

Der Klammeraffen-Befehl ist sehr nützlich für den Diskettenanwender. Eine Kurzübersicht finden Sie in Bild 13.

Befehl:		<@>
Syntax:		<@>(Gerätenummer)
	oder	<@>(Gerätenummer), Befehl
	oder	<@>(Gerätenummer), $(Optionen)
Wirkung:		Anzeigen des Disk-Status, wenn kein Komma auf Klammeraffe und evtl. Gerätenummer folgt.
		Senden eines Befehls an die Floppy, wenn Befehl folgt.
		Anzeige des Directory unter Berücksichtigung der Optionen, falls Disk-Befehl mit "$" beginnt.
englisches Wort:		-

Bild 13. Kurzübersicht zum Befehl @

Oft interessiert man sich nur für einen Teil des gesamten Directories. Das Floppylaufwerk ist in der Lage, einzelne Teile des Directories an den Computer zu senden, wenn es ihr nur mitgeteilt wird. Das »$« hinter dem Klammeraffen sendet der Monitor als Filename zum Floppylaufwerk. Dann arbeitet er wie der Basic-Befehl CATALOG, beziehungsweise DIRECTORY.

Bei diesen Befehlen lassen sich auch Selektierungen vornehmen, wenn dem Befehl ein entsprechender Filename, der natürlich mit »$« beginnen muß, folgt:

DIRECTORY"$:*=PRG" in Basic entspricht dem Monitor-Kommando: @,$:*=PRG.

Nun zum Aufbau des Directory-Ausschnitt-Filenamens.

Auf das »$« sollte ein Doppelpunkt »:« folgen. Danach wiederum kann ein Filename stehen. In diesem Filenamen dürfen dann Joker vorkommen (siehe Floppy-Handbuch).

»$:FILENAME« ist also die Anweisung, um den Directory-Eintrag des Programms FILENAME mitsamt Kopfzeile und »BLOCKS FREE«-Angabe auf den Bildschirm zu holen.

»$:FILE*« listet alle Files, deren Namen mit »FILE« beginnen.

»$:F??E« listet alle Files, deren erster Buchstabe im Filenamen F und deren vierter Buchstabe ein E ist. Damit erst gar nicht Mißverständnisse aufkommen:

$ ist kein Befehl an das Diskettenlaufwerk, den man über OPEN und PRINT# sendet und auch kein Monitor- oder Basic-Befehl. »$:FILENAME« allein ergibt nur einen Syntax Error oder das Monitor-Fragezeichen. Erst CATALOG"$:FILENAME" ist ein gültiger Basic- und @,$:FILENAME ein gültiger Monitor-Befehl.

Nun wieder zu den Anwendungen des Jokers:

Auch die Kombination ist möglich. »$:F?L*« listet alle Files, die folgende Bedingungen erfüllen:

1. Das erste Zeichen im Filenamen ist ein F
2. Das dritte Zeichen im Filenamen ist ein L
3. Der Filename besteht aus mindestens drei Zeichen.

Folgt nun der Angabe des Filenamens noch ein »=« und darauf der Filetyp, kann man auch nur bestimmte Files zulassen:

@,$:F?L*=SEQ

listet alle Files, die die oben genannten drei Bedingungen erfüllen und noch eine vierte:

4. Das File muß ein SEQ-File (sequentielles File) sein.

Nun verstehen wir auch, warum »@,$:*=PRG« alle Programmfiles auflistet, denn aus dem Filenamen »$:*=PRG« ergeben sich folgende Bedingungen:

1. Der Filename muß mindestens aus einem Zeichen bestehen. Diese Bedingung erfüllt jedes File, also können wir sie weglassen.
2. Es muß sich um ein Programm-File handeln.

Letztendlich muß also nur die zweite Bedingung (Programm-File) erfüllt sein.

Wir haben uns jetzt etwas ausführlich mit den Anwendungsmöglichkeiten der Directory-Ausgabe auseinandergesetzt, aber es hat sich gelohnt: Sie können sich diese Kennt-

nisse auch für die Basic-Befehle CATALOG und DIRECTORY zunutze machen!

Der Speicher-Befehl: S

Nachdem wir uns im letzten Abschnitt mit der Diskettenverwaltung befaßt haben, sollen uns nun die Ein-/Ausgabemöglichkeiten mit dem Monitor eine Weile beschäftigen, denn wenn man ein Programm erstellt, geändert oder getestet hat (für alle diese Anwendungen nimmt man ja den Monitor her), will man es meist auch speichern.

Der Befehl zum Speichern heißt S und ist die Abkürzung des englischen Wortes »to save«, das mit »speichern« zu übersetzen ist.

Die Syntax ist folgende:

S "FILENAME", Gerät, Anfangsadresse, Endadresse+1

Der Filename darf maximal 16 Zeichen lang sein (wie bei den Basic-Befehlen DSAVE und BSAVE zum Speichern). Die Geräteadresse wird als Byte-Wert (zum Beispiel »08« für Diskettenlaufwerk) angegeben. Anfangs- und Endadresse geben den Bereich an, der gespeichert werden soll. Dabei ist jedoch zu beachten, daß die Endadresse nicht mitgespeichert wird. Daher heißt es in der Kurzübersicht (Bild 14) auch »Endadresse+1«.

Wenn man also die Endadresse um eins erhöht, ist es möglich, den gesamten Bereich (auch das letzte Byte) zu speichern. Leider kann man dem Monitor nicht die Endadresse in Form von »$1F05+1« mitteilen, sondern muß die Addition selbst durchführen. Die Endadresse Ihres Programms dürfte Ihnen in der Regel bekannt sein, so daß die Addition nicht weiter schwerfallen sollte.

Nun aber wieder zum S-Befehl.

Der Befehl speichert den angegebenen Speicherbereich (die Speicherbank wird in den Adressen angegeben) auf Diskette ab. Hierfür gibt es aber doch schon den Basic-Befehl BSAVE, oder?

Das ist richtig. Der BSAVE-Befehl entspricht in der Wirkung dem S-Befehl des Monitors. Andererseits ist es aber gut, daß auch der Monitor einen Speicher-Befehl zur Verfügung hat, denn oft wurde der Speicher des Computers beim Arbeiten mit dem Monitor so verändert, daß ein X-Befehl zum erneuten BREAK (also Neustart des Monitors) oder gar zum Ausstieg, aber nicht zum Sprung ins Basic führt. In einem solchen Fall besteht die einzige Hoffnung im Monitor selbst, und Sie können das fehlerhafte oder fehlerhaft unterbrochene Programm wenigstens noch SAVEn.

Im übrigen kann der S-Befehl – wie alle anderen I/O-Befehle des Monitors auch (I/O-Befehle sind Befehle wie Klammeraffe oder S, die sich auf Eingabe/Ausgabe = Input/Output = I/O beziehen), zu einer Fehlermeldung führen, die das Betriebssystem erzeugt. Diese Meldung heißt »I/O ERROR #X«, wobei X eine Nummer ist, die über den Fehler Auskunft gibt. Ein gängiger Fehler ist »I/O ERROR #5«, was der Bedeutung der Basic-Fehlermeldung »DEVICE NOT PRESENT« entspricht.

In Bild 15 finden Sie die Bedeutung der einzelnen I/O-ERROR-Meldungen, die übrigens auch beim Klammeraffen-Befehl vorkommen können.

Es gibt noch einen weiteren Basic-Befehl zum Speichern, nämlich SAVE beziehungsweise DSAVE. Dieser speichert – wie BSAVE oder der Monitor-Befehl S – auch nur einen Bereich ab und zwar den, in dem das Basic-Programm abgelegt ist. Diesen Bereich muß er natürlich ausfindig machen, denn die Speicherroutine des Betriebssystems benötigt Anfangs- und Endadresse+1 eines Speicherbereichs.

Genau diese Informationen holt sich der Interpreter dann aus den Adressen $2D/$2E (Zeiger auf Anfangsadresse des Basic-Programms im Speicher) und $1210/$1211 (Zeiger

Befehl:	S
Syntax:	S "NAME",Gerät,Anfangsadresse Endadresse
Wirkung:	speichert File unter NAME auf das angegebene Gerät von Anfangsadresse bis Endadresse.
englisches Wort:	Save (retten=> speichern)

Bild 14. Kurzübersicht zum Befehl S

Fehlernummer	entsprechende Meldung in Basic
0	BREAK
1	TOO MANY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
6	NOT INPUT FILE
7	NOT OUTPUT FILE
8	MISSING FILE NAME
9	ILLEGAL DEVICE NUMBER

Bild 15. Übersicht über die I/O-Fehlermeldungen

auf Endadresse+1 des Basic-Programms im Speicher). Da wir das Auslesen dieser Zeiger mittels Monitor oder Basic schon im Abschnitt über den M-Befehl besprochen haben, wäre es theoretisch möglich, daß Sie die Zeiger auslesen und mit dem Monitor bearbeitete Basic-Programme dann über BSAVE oder S speichern. Dies ist aber eigentlich nur dann ratsam, wenn Sie aus dem Monitor heraus nicht mit X ins Basic zurückkehren können (zum Beispiel wegen Fehleingabe). Wann immer es geht, ist es viel einfacher, mit X ins Basic zu springen und das Basic-Programm – sofern es auch über LOAD beziehungsweise DLOAD geladen wurde – mittels SAVE beziehungsweise DSAVE zu speichern.

Ob es sich bei dem auf diese Weise gespeicherten Programm um ein Basic-Programm handelt, ist nicht von Interesse. Bedingung ist nur, daß das Programm

- über die Basic-Befehle DLOAD"NAME" oder LOAD"NAME",Gerät (nicht LOAD"NAME",Gerät,1) geladen wurde und
- die Zeiger $2D/$2E beziehungsweise $1210/$1211 auf Anfangs- beziehungsweise Endadresse des Basic-Programms zeigen (also diese Zeiger nicht manipuliert wurden).

Den Monitor-Befehl S setzt man in der Regel zum Speichern einzelner Bereiche (zum Beispiel Maschinenroutinen) ein.

Der Lade-Befehl: L

So wie S der Befehl zum Speichern ist, gibt es L als Befehl für »to load« (laden) zum Laden von Programmen.

Der L-Befehl hat zunächst einmal folgende Syntax:

L "FILENAME",Gerät,Ladeadresse

Das Beispiel »L "BEISPIEL",08,1C000« lädt das Programm »BEISPIEL« (Filename wie bei S maximal 16 Zeichen) von Gerät 8 (Diskettenlaufwerk) in Bank 1 ab Adresse $C000.

So wie der Monitor-Befehl S auch durch den Basic-Befehl BSAVE ersetzt werden kann, können wir dieses Beispiel folgendermaßen mittels BLOAD in Basic formulieren:

```
BLOAD"BEISPIEL",D0,U8,ON B1,P(DEC("C000"))
```

Der einzige Vorteil des Monitor-Befehls L gegenüber dem BLOAD-Befehl, beziehungsweise des BSAVE-Befehls gegenüber dem Monitor-Befehl S ist, daß bei den Monitor-

Befehlen die Geräteadresse frei wählbar ist (also auch Datasette möglich ist), während die Basic-Befehle BLOAD und BSAVE nur auf den IEC-Bus zugreifen (effektiv kann man also nur das Diskettenlaufwerk verwenden). Ob diese zusätzliche Möglichkeit der Monitor-Befehle tatsächlich ein Vorteil ist, muß dahingestellt bleiben, denn die Datasette ist wirklich kein angemessenes Speichergerät für den C128.

Der Befehl L kennt noch zwei weitere Formen der Syntax, denn man kann Ladeadresse oder Ladeadresse und Gerätenummer entfallen lassen. Dann werden sogenannte Default-Werte eingesetzt. Default-Werte sind Werte, die bei der Übergabe von Parametern dann eingesetzt werden, wenn kein Wert geliefert wird.

»L "BEISPIEL",08« lädt das Programm »BEISPIEL« von Gerät 8 (Diskettenlaufwerk) an die Anfangsadresse, die im Programm-File vermerkt ist. Mit dieser Anfangsadresse werden wir uns noch näher befassen, denn diese ist genauso wichtig für ein File wie Blockzahl, Programmlänge oder Filetyp, wird aber leider nicht im Directory angezeigt. Nur soviel sei zunächst gesagt: In PRG-Files wird auch die Anfangsadresse beim Speichern des Files mitgespeichert. Falls wir diese beim Monitor-Befehl L oder beim Basic-Befehl BLOAD weglassen, wird die im Programm vermerkte Anfangsadresse als Ladeadresse verwendet.

Gleiches geschieht beim Basic-Befehl LOAD"BEISPIEL",8,1. Diesen wollen wir jedoch übergehen, denn zum Laden von Maschinenprogrammen/Speicherbereichen sollte man auf diese Variante verzichten und lieber auf BLOAD beziehungsweise den Monitor-Befehl L zugreifen.

Die letzte Syntax-Variante von L ist
L "BEISPIEL"

In diesem Fall wird das File von Datasette geladen und zwar an die im Programm vermerkte Anfangsadresse, da der C128 als Default-Gerät die Datasette gespeichert hat.

Die gesamte Syntax von L finden Sie noch einmal in Bild 16 als Kurzübersicht zusammengefaßt.

Für Programme, die ohnehin Basic-Programme sind oder zumindest eine Basic-Zeile haben und am Basic-Start beginnen, ist der Basic-Befehl LOAD beziehungsweise DLOAD eindeutig vorzuziehen. Dieser bietet nämlich den Vorteil, daß das Programm hinterher nicht mit dem Monitor oder über BLOAD gespeichert werden muß – wo man ja Anfangs- und Endadresse selbst bestimmen muß –, sondern daß man das

Befehl:	L
Syntax:	L "NAME"(,Gerätenummer)(,Ladeadresse)
Wirkung:	lädt File "NAME" vom angegebenen Gerät (falls weggelassen: #1=Datasette) an die Adresse, ab der es gespeichert wurde, oder – falls angegeben – an die gewünschte Ladeadresse.
englisches Wort:	Load (laden)

Bild 16. Kurzübersicht zum Befehl L

Befehl:	V
Syntax:	V "NAME"(,Gerätenummer)(,Ladeadresse)
Wirkung:	vergleicht aktuellen Speicher – falls angegeben: von der Ladeadresse an – mit dem File "NAME" auf dem angegebenen Gerät (falls weggelassen: 1).
englisches Wort:	Verify (überprüfen)

Bild 17. Kurzübersicht zum Befehl V

Programm später über SAVE beziehungsweise DSAVE (siehe letzten Abschnitt) speichern kann.

Fassen wir noch einmal kurz zusammen: Der BLOAD- beziehungsweise Monitor-L-Befehl ist nur beim Laden von Maschinenprogrammen angebracht. In der Wirkungsweise besteht kein Unterschied, der L-Befehl des Monitors erlaubt jedoch auch das Laden von Datasette. Der LOAD- beziehungsweise DLOAD-Befehl in Basic ist zur Bearbeitung von Basic-Programmen oder solchen, die in einigen Punkten wie Basic-Programme zu handhaben sind, besser geeignet.

Im nun folgenden restlichen Teil dieses Abschnitts wollen wir uns noch mit den Unterschieden zwischen dem Laden an eine angegebene Ladeadresse und dem Laden an die im Programm enthaltene Anfangsadresse befassen. Insbesondere werden wir noch die dafür verwendeten Fachbegriffe kennenlernen.

Absolutes Laden

Absolut laden heißt, daß ein Programm an die Adresse geladen wird, die beim Speichern als Anfangsadresse verwendet wurde.

Beim Weglassen der Ladeadresse beim Basic-Befehl BLOAD beziehungsweise beim Monitor-Kommando L wird absolut geladen: Das Betriebssystem holt sich zuerst die Anfangsadresse, dann wird an diese geladen. Das absolute Laden ist vor allem bei Maschinenprogrammen angebracht.

Relatives Laden

Relativ laden heißt, daß ein Programm an eine vorgegebene Adresse geladen wird, und zwar unabhängig von der Adresse, ab der es gespeichert wurde.

Bei Angabe der Ladeadresse beim Basic-Befehl BLOAD beziehungsweise beim Monitor-Komando L wird relativ geladen: Das Betriebssystem lädt das Programm an die angegebene Anfangsadresse.

Da Basic-Programme ziemlich unabhängig vom Speicherbereich sind, in dem sie sich gerade befinden (Beweis ist die Möglichkeit des Verschiebens der Basic-Programme im Abschnitt über den T-Befehl, beziehungsweise beim Einschalten der hochauflösenden Graphik, wo das aktuelle Basic-Programm nach $4001 verschoben wird), ist der Basic-Befehl LOAD beziehungsweise DLOAD darauf eingerichtet, das Programm an eine Ladeadresse zu laden. Sie werden sich vielleicht fragen, woher diese Ladeadresse kommt, wenn man doch bei LOAD"BEISPIEL",8 oder DLOAD"BEISPIEL" keine Ladeadresse angibt. Die Frage ist berechtigt, aber um die Ladeadresse müssen wir uns keine Sorgen machen. Diese Arbeit nimmt uns der Interpreter ab, der beim Laden über DLOAD (beziehungsweise LOAD ohne Angabe der Sekundäradresse) die Adresse als Ladeadresse einsetzt, die im Zeiger $2D/$2E als Anfangsadresse des aktuellen Basic-Programms im Speicher abgelegt ist.

Da Maschinenprogramme meist nur in einem Bereich lauffähig sind, kommt für Sie relatives Laden selten in Betracht.

Der Überprüf-Befehl: V

Zum »Triumvirat« der I/O-Befehle LOAD, SAVE und VERIFY fehlt uns nur noch letzterer. Dieser Befehl heißt beim TEDMON, wie nicht anders zu erwarten war, V als Abkürzung für »to verify« (englisches Wort für »nachprüfen«).

Die Syntax ist die des L-Befehls (die Routine des Betriebssystems für L beziehungsweise V ist sogar ein und dieselbe) und kann noch einmal der Kurzübersicht (Bild 17) entnom-

men werden. Der Unterschied in der Funktionsweise ist der, daß das Programm nicht in den Speicher geladen (wie es bei L der Fall wäre), sondern mit den entsprechenden Speicherinhalten verglichen wird.

Falls keine Übereinstimmung besteht, erscheint die Meldung »VERIFYING ERROR«, ansonsten nur »VERIFYING«.

Da mit den aktuellen Speicherinhalten Vergleiche durchgeführt werden, ist es sogar möglich, relativ, beziehungsweise absolut, zu vergleichen.

Der V-Befehl ist eigentlich nur deshalb interessant, weil es keinen BVERIFY-Befehl im Basic gibt. Um nur die Lauffähigkeit der Floppy 1571 zu testen, ist der V-Befehl überflüssig, denn die Floppy 1571 führt selbständig ein Verify durch (dabei wird allerdings nicht mit dem Speicher, sondern mit dem Puffer der Floppy-Station verglichen).

Nun haben wir alle I/O-Befehle des Monitors (@, L, S, V) besprochen. Der folgende Abschnitt wird sich intensiv mit dem Thema Ein-/Ausgabe (I/O) befassen.

Zusätzliche Erklärungen zur Ein-/Ausgabe mit TEDMON

Die absolute Anfangsadresse eines Programms ist die Adresse, ab der das Programm gespeichert wurde. Beim absoluten Laden ist diese Adresse der Ersatz der Ladeadresse beim Relativladen.

Bei einem Diskettenfile enthalten die beiden ersten Bytes die absolute Anfangsadresse:
Erstes Byte des Files: LB der absoluten Anfangsadresse
Zweites Byte des Files: HB der absoluten Anfangsadresse

Auf der 1571-Test/Demo-Diskette befindet sich übrigens ein interessantes Programm zum Auslesen und Manipulieren der Anfangsadresse. Es trägt den Filenamen »LOAD ADDRESS«, übersetzt also »Ladeadresse«.

Bei vielen Maschinenprogrammen ist die absolute Anfangsadresse gleich der Startadresse. Das heißt, wenn Sie ein Pro-

```
100 REM *********************************
110 REM *                               *
120 REM *        ANFANGSADRESSE         *
130 REM *                               *
140 REM *        UND ENDADRESSE         *
150 REM *                               *
160 REM *   EINES DISKETTENPROGRAMMS    *
170 REM *                               *
180 REM *          ERMITTELN.           *
190 REM *                               *
200 REM *********************************
210 REM *                               *
220 REM *  1985/86 BY FLORIAN MUELLER   *
230 REM *                               *
240 REM *********************************
250 :
260 IFPEEK(215)<>128THEN PRINT"NUR IM 80-ZEICHE
N-MODUS LAUFFAEHIG":STOP
270 FAST:SCNCLR:W$="#####"
280 PRINTCHR$(27)"O":PRINT"BESTIMMUNG VON ANFAN
GS- UND ENDADRESSE"CHR$(13)
290 INPUT"FILENAME";N$:IF N$="$" THEN PRINT:DIR
ECTORY:PRINT:GOTO 280
300 OPEN15,8,15,"IO"
310 IF DS<>OTHEN340
320 OPEN1,8,3,N$+",P,R"
330 IFDS=0THEN410
340 PRINT:PRINTCHR$(15)"DISKFEHLER: ";DS$:PRINT
350 CLOSE1:CLOSE15:GOTO290
360 :
370 REM ***************************
380 REM * INFORMATIONEN AUSGEBEN *
390 REM ***************************
400 :
410 SCNCLR:PRINT:PRINT"INFORMATIONEN FUER "CHR$
(34);N$;CHR$(34)
420 PRINT
430 :
440 REM ******************
450 REM * ANFANGSADRESSE *
460 REM ******************
470 :
480 PRINT:PRINT"ANFANGSADRESSE:",,
490 GET#1,A$:GET#1,B$
500 IFA$=""THENA$=CHR$(0)
510 IFB$=""THENB$=CHR$(0)
520 A=ASC(A$)+256*ASC(B$)
530 PRINT USING W$;A;:PRINT" = $";HEX$(A)
540 CLOSE1
550 :
560 REM ***********************
570 REM * ANZAHL DER BLOECKE *
580 REM ***********************
590 :
600 PRINT:PRINT"ANZAHL DER BLOECKE AUF DISK:",
610 OPEN1,8,0,"$:"+N$
620 GET#1,A$,B$
630 GET#1,A$,B$
640 GET#1,A$,B$
650 GET#1,B$:IFST<>0THEN840
660 IFB$<>CHR$(34)THEN650
670 DO:GET#1,B$:LOOP WHILE B$<>CHR$(34)
680 DO:GET#1,B$:LOOP WHILE B$=CHR$(32)
690 DO:GET#1,B$:LOOP WHILE B$<>""
700 GET#1,A$,B$,A$,B$
710 IFA$=""THENA$=CHR$(0)
720 IFB$=""THENB$=CHR$(0)
730 B=ASC(A$)+256*ASC(B$)
740 PRINT USING W$;B;:PRINT" = $";HEX$(B)
750 CLOSE1
760 :
770 REM ************************
780 REM * MAXIMALE ENDADRESSE *
790 REM ************************
800 :
810 PRINT:PRINT"MAXIMALE ENDADRESSE:",
820 E=A+254*B-2
830 PRINT USING W$;E;:PRINT" = $";HEX$(E)
840 CLOSE1:CLOSE15
850 PRINT:PRINTCHR$(15)"TASTE DRUECKEN (G FUER
GENAUE ENDADRESSE/D FUER DRUCKERAUSGABE DER DAT
EN)";
860 GETKEY GK$:IF GK$="D"THENEE=E:M=1:GOTO1070:
ELSE IF GK$<>"G"THEN RUN
870 :
880 REM **********************
890 REM * GENAUE ENDADRESSE *
900 REM **********************
910 :
920 M=0
930 PRINTCHR$(27)"D":PRINTCHR$(145)"GENAUE ENDA
DRESSE FUER 'S':",
940 EE=A:OPEN 1,8,0,N$
950 DO UNTIL ST AND 64
960 :GET#1,A$:EE=EE+1
970 LOOP:EE=EE-1
980 PRINTUSING W$;EE;:PRINT" = $";HEX$(EE)
990 CLOSE1
1000 PRINT:PRINTCHR$(15)"TASTE DRUECKEN (D FUER
 DRUCKERAUSGABE DER DATEN)";
1010 GETKEY GK$:IF GK$<>"D"THEN RUN
1020 :
1030 REM ******************
1040 REM * DRUCKERAUSGABE *
1050 REM ******************
1060 :
1070 V$="##############################":REM 30
 * #
1080 PRINTCHR$(27)"D"
1090 OPEN 4,4
1100 PRINT#4,USING V$;"FILENAME:";:PRINT#4,N$
1110 PRINT#4,USING V$;"ANFANGSADRESSE:";:PRINT#
4,USING W$;A;:PRINT#4," = $";HEX$(A)
1120 PRINT#4,USING V$;"ENDADRESSE:";:PRINT#4,US
ING W$;E;:PRINT#4," = $";HEX$(EE);:IF M=1 THEN
PRINT#4," (MAXIMAL)":ELSE PRINT#4," (FUER 'S'-B
EFEHL)"
1130 PRINT#4,USING V$;"LAENGE IN DISK-BLOECKEN:
";:PRINT#4,USING W$;B;:PRINT#4," = $";HEX$(B)
1140 CLOSE4
1150 IF M=1 THEN 850:ELSE GOTO 1000
```

Listing 6. Ermitteln von Anfangs- und Endadresse eines Diskettenprogramms

gramm an die Adresse 49152 laden, wird es unter Umständen auch mit SYS 49152 gestartet. Hierfür kann jedoch keinerlei Garantie übernommen werden.

Der Unterschied zwischen Anfangs- und Startadresse ist folgender: Die Anfangsadresse beschreibt nur die »geographische« Lage im Speicher, sagt aber nichts über den Start des Programms aus. Die Startadresse hingegen gibt keine Auskunft darüber, wo das Programm im Speicher beginnt, sagt uns aber, wo man das Programm startet. In der Literatur werden diese Begriffe oft durcheinandergewürfelt.

Ermittlung von Anfangs- und Endadresse

Für die Anwendung der Befehle L, S und V ist es sehr wichtig zu wissen, wo im Speicher das Programm liegt, das man bearbeiten möchte. Dazu hilft uns ein Programm, das zwar das längste, aber sicher auch nützlichste Listing dieses Kurses ist: Listing 6. Dieses Listing sollten Sie unbedingt abtippen, denn die geringe Abtipparbeit erspart Ihnen hinterher viel Mühe. Das Programm arbeitet übrigens nur mit einem Diskettenlaufwerk. Wenn Sie bei der Eingabe des Filenamens »$« eingeben und <RETURN> drücken, wird das Directory ausgegeben.

Bei anderen Filenamen wird das entsprechende Programm untersucht. Zunächst errechnet Listing 6 die Anfangsadresse wie beschrieben, dann wird aus dem Directory die Block-Anzahl herausgesucht und aus dieser Angabe die maximal mögliche Endadresse errechnet. Voraussetzung für die Richtigkeit dieser Angabe ist, daß die Block-Anzahl im Directory nicht manipuliert worden ist.

Die Endadressen-Angaben erfolgen übrigens immer um eins erhöht, so daß man sie direkt für den S-Befehl übernehmen kann. Wenn es auf ein paar Bytes hin oder her nicht ankommt, reicht die Angabe der maximalen Endadresse. Die exakte Endadresse, die auf Wunsch auch bestimmt wird, kann man direkt beim S-Befehl einsetzen.

Außerdem ermöglicht das Programm eine Druckerausgabe, so daß Sie sich einen Katalog mit den genauen Daten Ihrer Programme leicht zusammenstellen können.

Im folgenden sei aber auch eine Methode zur Auffindung der exakten Endadresse ohne Hilfsprogramm vorgestellt. Wir numerieren die Schritte mit römischen Ziffern.

I. Speicher mittels F mit einem Füll-Byte, das relativ selten ist, füllen. Beispiel: F 1C01 FDFF AB

$AB hat sich als Füll-Byte bewährt, natürlich sind auch ähnliche Zahlen wie $AA oder $FD möglich. $00 oder $FF sollte man nicht verwenden, da diese Bytes (insbesondere $00) oft am Ende eines Programms stehen – an späterer Stelle würden Sie dann merken, daß sich in so einem Fall nicht die exakte Endadresse bestimmen ließe.

II. Programm mittels L laden. Beispiel: L "NAME",08,4000

Nun sieht unser Speicher folgendermaßen aus:

A) *** Speicherbeginn-Programmbeginn ***
B) *** Programmbeginn (im Beispiel $4000)***
C) *** Programmcode ***
D) *** Programmende (letztes Byte) ***
E) *** Füll-Bytes (im Beispiel $AB-Bytes) ***
F) *** Speicherende ***

Die Endadresse ist nun der Punkt D, also die letzte Speicherzelle vor dem Bereich, der nur mit Füll-Bytes gefüllt ist. Da für den S-Befehl die Endadresse immer um eins erhöht werden muß, können wir gleich Punkt E nehmen (Anfangsadresse des Bereichs, der nur mit Füll-Bytes belegt ist).

Vor dem Laden (Punkt II) war der gesamte Speicher ab Stelle B mit Füll-Bytes belegt; durch das Laden wurde ein Teil der Füll-Bytes vom Programmcode ersetzt. Da es sich beim geladenen Programm in der Regel jedoch nicht um den ganzen Speicher handelt, ist noch der Bereich ab Stelle E mit den Füll-Bytes belegt. Folglich ist die erste Adresse, ab der – bis Speicherende – nur noch Füll-Bytes stehen, die Endadresse des Programms plus eins ist die tatsächliche Endadresse des Programms für den S- beziehungsweise BSAVE-Befehl.

Im nächsten Schritt werden wir diese Endadresse nun ermitteln.

III. Grobeingrenzung des Bereichs, der nur aus Füll-Bytes besteht, mittels H.

Das Problem hierbei ist, daß der Wert des Füll-Bytes – wenn auch nur selten – sicher ebenfalls im Programmcode vorhanden ist. Wenn wir nun mit H den Speicher ab der Anfangsadresse des Programmcodes nach dem Füll-Byte durchsuchen lassen (im Beispiel: H 4000 FDFF AB), so findet der Monitor ab und zu das Füll-Byte. Ab einem gewissen Punkt, der mit ziemlicher Sicherheit die Endadresse plus eins ist, findet der Computer eine Adresse nach der anderen. Wenn dies der Fall ist, sollten Sie unbedingt die Ausgabe mit <RUN/STOP> unterbrechen. Ungefähr können Sie jetzt an den gefundenen Adressen erkennen, ab welcher Stelle nur noch Füll-Bytes stehen: ab der erstgenannten Adresse, von der an nur noch Füll-Bytes stehen.

Meist haben Sie nun schon die Adresse ausfindig gemacht, aber wenn Sie noch unsicher sind, gibt es die »Feinbestimmung«.

Zeigen Sie den Bereich an, der Ihrer Meinung nach in Frage kommt, und suchen Sie genau die Stelle heraus, ab der nur Füll-Bytes stehen.

Das war dann das Verfahren ohne Hilfsprogramm. Sie werden es vor allem auf anderen Computern, auf denen das Hilfsprogramm nicht läuft, fast unverändert verwenden können. Beim Arbeiten mit dem C 128 sollten Sie doch das Hilfsprogramm verwenden – wie gesagt, die Abtipparbeit, die sich ohnehin in Grenzen hält, wird sich mehrfach lohnen.

Unser Drei-Schritte-Verfahren kommt vor allem bei der Arbeit mit der Datasette zur Anwendung, es hat aber auch gezeigt, wieviel man mit dem Monitor ermitteln kann, was sonst nicht möglich ist.

Dafür, daß Sie sich so lange mit der Lade/Speicher-Theorie abgemüht haben, sollen Sie durch einen kleinen Trick belohnt werden.

Leider existiert im Basic 7.0 kein Befehl zum Speichern der Bitmap der hochauflösenden Grafik (im 40-Zeichen-Modus natürlich).

Wenn man nun weiß, daß die Grafik bei $2000 (#8192) beginnt und genau 64000 Bit (1 Bit pro Pixel), also 8000 Byte, umfaßt, kann man die Endadresse auch ausrechnen: Endadresse. Bitmap = $2000 + #8000 = $3F40 = #16192

Da zur Endadresse 1 addiert werden muß, kann man im Monitor die Grafik-Bitmap folgendermaßen speichern:

S "BITMAP",08,2000 3F41

Über Basic geht dies mit

BSAVE "BITMAP",P8192 TO P16193

Die gespeicherte Bitmap läßt sich leicht über BLOAD "BITMAP" oder L "BITMAP",08 (im Monitor) laden.

Druckerausgabe mit dem Monitor

Bis jetzt haben wir Ihnen, um Sie nicht mit zu viel Informationen zu belasten, verschwiegen, wie man die Ausgaben des TEDMON auf den Drucker leitet.

Als Beispiel wollen wir den Bereich $FC00 bis $FC30 in Bank 15 ($F) auf den Drucker dumpen lassen. Dazu gehen wir folgendermaßen vor:

```
OPEN 4,4:CMD 4:MONITOR
```

Sie sehen nun, wie die Monitor-Einschaltmeldung auf dem Drucker ausgegeben wird.

Ferner blinkt der Cursor – wie normal.

```
M FFC00 FFC30
```

Sie sehen nun, wie eine Dump-Zeile nach der anderen ausgegeben wird – auf dem Drucker!

Aber Vorsicht: Die letzte Zeile befindet sich wahrscheinlich noch im Drucker-Puffer und nicht auf dem Papier. Wenn Sie einen weiteren Ausgabe-Befehl ausführen lassen, wird zuerst diese letzte Zeile ausgegeben und dann die neue Ausgabe. Wenn Sie aber den Monitor verlassen wollen, lassen Sie bitte Ihren Drucker eingeschaltet und geben Sie zuerst X ein. Dann schließen Sie noch über

```
PRINT#4: CLOSE4
```

den Druckerkanal. Wenn Sie diese letzten Befehle nicht eingeben, wird unter Umständen eine Ausgabe-Zeile verschluckt!

Die Druckerausgabe ist vor allem dann sinnvoll, wenn man Maschinenprogramme disassembliert (?) oder dumpt. Falls Ihnen noch nicht bekannt ist, was Disassemblieren, Assemblieren oder Registeranzeige sind, so erfahren Sie alles darüber in den folgenden Abschnitten.

TEDMON und Maschinensprache

Nachdem wir uns bislang nur mit den Anwendungsmöglichkeiten des TEDMON auf Basic-Programme befaßt haben, sollen nun auch die Maschinenprogrammierer auf ihre Kosten kommen.

Die bisher erklärten Befehle haben als Basis den M-Befehl, denn alle stehen in enger Verbindung mit ihm. Bei den Ein/Ausgabe-Befehlen, die für jede Art Anwendung des Monitors interessant sind und auch entsprechend besprochen wurden, kann man das allerdings nicht sagen.

Im folgenden werden wir einige Befehle, die speziell für die Anwendung des TEDMON auf Maschinenprogramme vorgesehen sind, besprechen.

Die Basis dieser Befehle ist wohl der A-Befehl, den wir allerdings nicht als ersten besprechen (M war ja auch nicht unser erster Befehl).

Den ersten Kontakt mit einer Monitor-Funktion für die Behandlung von Maschinenprogrammen haben wir schon beim Starten des Monitors über MONITOR. Es handelt sich hierbei um die Registeranzeige (englisch »register display«).

Diese haben wir zunächst übergangen, hier soll aber jedes Detail dieser Registeranzeige besprochen werden.

Wenn der Monitor über den Assembler-Befehl BRK aktiviert wird, so merkt sich der Monitor den Zustand der einzelnen Register.

Diese werden dann, nachdem sie zwischengespeichert worden sind, automatisch angezeigt:

```
   PC  SR AC XR YR SP - Erklärungszeile
;FB000 00 00 00 00 F8 - Werte der Register
```

Diese Meldung erhalten Sie beim Start über den Basic-Befehl MONITOR, da bei diesem der TEDMON von $B000 in Bank $F angesprungen wird. Daher steht unter PC (PC steht für »program counter« oder deutsch »Programmzähler«) die Adresse $FB000. An dieser Adresse ist beim Start über den Maschinenbefehl BRK (Opcode: $00) erkennbar, von wo aus der BRK erfolgte. Dazu muß man allerdings vom PC-Wert 2 subtrahieren. Außerdem ist zu beachten, daß als Speicherbank bei PC immer $F angegeben wird. Der Grund dafür ist der, daß der Prozessor bei einem BRK wie bei anderen, ähnlichen Impulsen (Reset, IRQ, NMI) immer $F als Speicherbank einstellt.

SR ist der Zustand des Statusregisters, das die einzelnen Statusbits enthält (N-, V-, B- ,D-, I-, Z- und C-Flag). Mit Hilfe der Befehle % und $ kann man nun einzelne Bits behandeln und das Ergebnis als ganzes Byte ins Hexadezimalformat umwandeln lassen.

Die drei darauffolgenden Zwei-Buchstaben-Abkürzungen stehen für Akku, X- und Y-Register:

AC = ACcumulator = Akkumulator
XR = X Register = X-Register
YR = Y Register = Y-Register

SP ist der Stapelzeiger (englisch »stack pointer«). $F8 ist der Initialisierungswert des Stapelzeigers. Bei dem Wert, der unter SP steht – also dem Stapelzeigerinhalt – kann es durchaus vorkommen, daß bei Ihnen andere Werte als $F8 stehen (zum Beispiel $F9). Bei Reset und gleichzeitigem Halten von <RUN/STOP> wird übrigens auch der Monitor angesprungen. Unter SP steht dann $FF (der Initialisierungswert des Stapelzeigers, den das Betriebssystem in der Resetbehandlung gibt).

In der Zeile unter den Zwei-Buchstaben-Abkürzungen, die nun erklärt sind (die Bedeutung der ausgeschriebenen Wörter sind einem Assembler-Lehrbuch zu entnehmen), stehen die einzelnen Inhalte der darüberstehenden Register im Hexadezimalformat.

Das Semikolon »;« am Anfang der Zeile, die die einzelnen Werte angibt, ist ein eigenständiger Befehl, der uns ein Ändern der Registerinhalte ermöglicht. Wie beim M-Befehl (dort ist der Änderungsbefehl das >) müssen wir dazu nur die alten Werte überschreiben. Soviel Informationen finden Sie in jeder guten Beschreibung, es muß aber eins unbedingt gesagt werden, um Mißverständnisse auszuräumen.

Die Register werden nicht in Wirklichkeit geändert! Wenn Sie also mit <RETURN> die Register übernehmen lassen, ändern sich nicht die echten Prozessorregister (da der Monitor ein Maschinenprogramm ist und somit dauernd die Register einsetzt und dabei natürlich ändert, wäre dies auch weder möglich noch sinnvoll), sondern nur die Zwischenspeicher der Prozessorregister.

Diese Zwischenspeicher werden beim Anspringen eines Maschinenprogramms über den G-Befehl, der noch besprochen wird, in die tatsächlichen Prozessorregister übernommen.

An späterer Stelle werden wir uns wieder mit der Registeranzeige und vor allem der Anwendung beschäftigen. Zunächst soll uns aber der Start des Monitors über den BRK-Befehl interessieren.

Der Start über den Assembler-Befehl BRK

Wenn der Prozessor bei der Abarbeitung eines Maschinenprogramms auf den Code $00 als Befehl trifft, interpretiert er diesen als BRK-Anweisung. BRK (Abkürzung für »BReaK«) ist das Mnemonic für den Befehl, der einen Abbruch des aktuellen Maschinenprogramms veranlaßt.

Eine Wirkung des BRK-Befehls ist die, daß der Programmzähler (PC) auf den Stapel gelegt wird. Dann springt der Prozessor über einen Vektor, der im ROM steht, zur BRK-Routine des Betriebssystems. Diese wiederum springt über den BRK-Vektor (auch Break-Vektor genannt) $0316/$0317 zu einer weiteren BRK-Routine ab $B003, die (fast) dem Monitor-Start gleichkommt.

In der Testphase eines Maschinenprogramms ist es also ganz interessant, an allen möglichen Stellen des Programms BRK-Befehle einzusetzen; wenn der Computer auf so einen BRK stößt, wird der Monitor gestartet und dank der automatischen Registeranzeige kann man bequem sehen, wie das Programm gearbeitet hat.

Der Disassemblier-Befehl: D

Bevor wir uns nun in die Assemblier- und Disassemblier-Befehle stürzen, müssen wir schnell noch die Begriffe klären.

Beide Formen der Programmbehandlung dienen der Maschinensprache-Programmierung. Maschinensprache ist die Sprache, die die CPU (»Central Processing Unit« = zentrale Prozessoreinheit) als Hauptsteuerungsbaustein unseres Computers versteht. Diese Maschinensprache besteht nur aus elektronischen Signalen, also 0- und 1-Bits. 8 Bit gehören auch hier fest zusammen (Byte). Ein Byte kann sowohl für einen Befehl (zum Beispiel LDA) als auch für einen Operand stehen. Wenn man Bytes hexadezimal darstellt, sieht »LDA #$20« folgendermaßen im Speicher aus: A9 20.

Mit LDA #$20, beispielsweise als ASCII-Code im Speicher abgelegt, kann der Computer ohne bestimmte Programme nichts anfangen. Er versteht zunächst nur %10101001 (ein- aus- ein- aus- ein- aus- aus- ein) als Anweisung zum Laden des Akkumulators. Wenn wir statt dessen A9 schreiben, so machen wir es damit für uns lesbarer. An der Tatsache, daß der Computer nur binär arbeitet, ändert dies nichts.

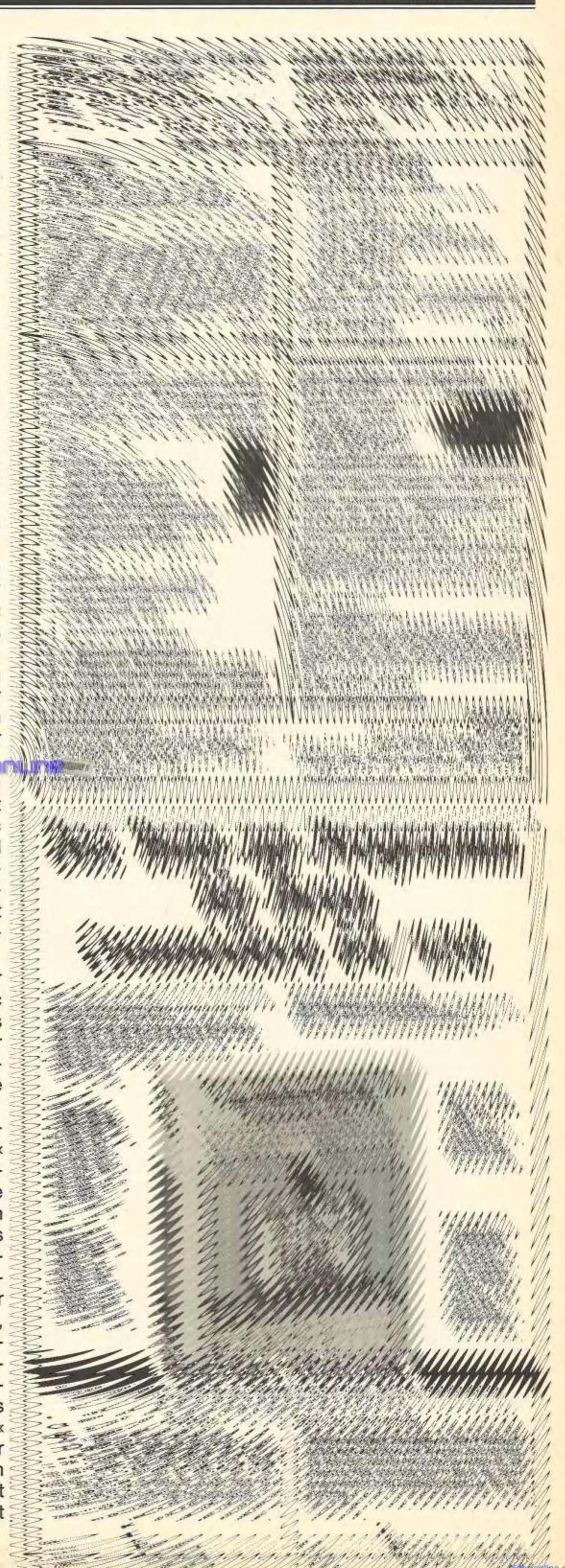
Am Rande sei hier nur erwähnt, daß der Computer genaugenommen auch kein Basic versteht. Das geht nur aufgrund eines sehr langen Maschinenprogramms, das fest im ROM eingebaut ist und »Basic-Interpreter« heißt, da es Basic-Befehle in die für die CPU verständliche Maschinensprache »übersetzt«.

Nun aber wieder zur Darstellung eines Maschinenprogramms: Auch wenn der erste Schritt getan ist, indem wir anhand von hexadezimalen Bytes das Programm »lesen« können, so ist es doch sehr schwer, A9 20 – wenn man es beim Memory-Dump erhält – als »lade Akku mit $20« zu verstehen. Zudem gibt es ja auch Befehle, die nicht nur einen Operand ($20), sondern gleich 2 (LB und HB) oder gar keinen haben, und dann müßte man zusätzlich zu unzähligen Opcodes auch die Länge der darauffolgenden Parameter wissen.

Kurzum, es ist nahezu unmöglich. Daher stellen wir diesen Befehl auch mit »LDA #$20« dar. Diese Schreibweise nennt man mnemonische Schreibweise, da »LDA« ein Mnemonic (das Wort »Mnemonic« kommt aus dem Griechischen und heißt »Gedächtnishilfe«) ist. LDA ist zwar nicht so schön verständlich wie ein Basic-Befehlswort (zum Beispiel COLLISION oder COLOR), aber immerhin besser als A9. LDA ist nämlich die Abkürzung eines sinnvollen englischen Ausdrucks: LoaD Accumulator.

Ein weiterer Vorteil der Mnemonics ist, daß man weniger Mnemonics als Opcodes benötigt, da auf ein Mnemonic mehrere Adressierungsarten entfallen können (wobei das Mnemonic jedesmal LDA heißt), zum Beispiel LDA $20 oder LDA #$20 oder LDA ($20),Y, wogegen jede neue Adressierungsart für ein und denselben Befehl einen neuen Opcode braucht.

Die Mnemonics sind also die eindeutig bessere Schreibweise eines Maschinenprogramms, da sie einer »Sprache« schon viel näher kommen als Nullen und Einsen oder Hex-Bytes. Der Haken dabei ist jedoch, daß der Computer wie gesagt mit den Mnemonics nichts anzufangen weiß. Natürlich könnte man sein Programm auf dem Papier in Mnemonics schreiben und dann in mühevoller Kleinarbeit ins hexadezimale Format umwandeln, so daß man es mit dem Monitor eingeben kann. In den Anfangsstadien des Computerns war dies auch so. Mittlerweile sind wir jedoch schon viel weiter, und da dieses Umwandeln eine reine Rechenaufgabe ist, lassen wir es vom Computer durchführen. Ein solches Umwandeln von Mnemonics ins Speicherformat nennt man »Assemblieren« (gesprochen: »äßemblieren«). Ein Programm, das diese Umwandlung durchführt, nennt man »Assembler« (gesprochen: »äßembler«). Im TEDMON ist ein solcher Assembler in Primitivstform eingebaut. Den entsprechenden Befehl lernen wir im nächsten Abschnitt kennen. Beim Wort Assembler bestehen, da man statt Maschinensprache oft



auch Assembler(-sprache) sagt, Verwechslungsmöglichkeiten, die der Sinnzusammenhang allerdings oft ausräumt.

Der dem »Assemblieren« entgegengesetzte Vorgang, nämlich das Umwandeln des Speicherformats eines Programms in mnemonische Schreibweise – zwecks besserer Lesbarkeit – heißt »disassemblieren« (sprich: »disäßemblieren«).

Der entsprechende Befehl heißt D (Abkürzung für »disassemble«). Die Syntax entspricht der des M-Befehls, mit dem Unterschied, daß statt eines Dumps ein Disassemblerlisting des entsprechenden Bereichs ausgegeben wird. Für das Weglassen einzelner Parameter bei »D Anfang Ende« gelten die gleichen Regeln wie bei M. Die Syntax von D ist noch einmal in Bild 18 zusammengefaßt.

Als Beispiel wollen wir ein Stück aus dem C128-ROM disassemblieren; geben Sie dazu bitte »D F4018 F4023« ein. Das Ergebnis sehen Sie auf Ihrem Bildschirm und in Bild 19.

Eine Disassemblerzeile hat das Format ».Adresse Hex-Bytes Mnemonic Operand«. Der Punkt ist eine andere Schreibweise für den Assemblier-Befehl und ermöglicht Änderungen (am Beispiel jedoch nicht, da wir Teile des ROMs disassembliert haben), worauf wir erst im nächsten Abschnitt näher eingehen wollen.

Die Adresse gibt wie bei M an, ab welcher Basisadresse disassembliert wurde. Die Hex-Bytes (minimal eines, maximal drei) sind das Hex-Dump der Bytes, die der Befehl umfaßt. Mnemonic und Operand sind dann schließlich die eigentliche Leistung des D-Befehls, denn die Hex-Bytes erscheinen nun in mnemonischer Schreibweise.

Die Fragezeichen in den Zeilen F4021/F4022 zeigen an, daß die Disassemblierroutine auf einen Opcode traf, der keinen Befehl vertritt. Es handelt sich in solchen Fällen entweder um reine Daten, die beispielsweise als ASCII-Codes abgelegt sind, oder – wie in diesem Fall – um reine Füll-Bytes.

Wie man auch aus F4023 ersehen kann, wird das Disassemblieren unmittelbar nach einem solchen undefinierten Byte fortgesetzt.

Meist handelt es sich bei undefinierten Bytes um Daten oder Füll-Bytes, es gibt jedoch auch eine dritte Möglichkeit, die nicht so bekannt ist: Undefinierte Bytes können auch undefinierte Opcodes sein!

Zusätzlich zu den Standard-6502-Befehlen kennt fast jeder 65xx-Prozessor zusätzliche Opcodes, die beim Bau des Prozessors unbeabsichtigt waren, aber aufgrund der Prozessor-Struktur verarbeitet werden und meistens auch ihre Wirkung haben. Diese Befehle sind zwar – da sie ja unbeabsichtig sind – von Prozessor zu Prozessor und von Version zu Version verschieden, aber werden gerne verwendet, um ein Programm unlesbar zu gestalten. Einige dieser Opcodes führen zum Absturz des Computers, andere entsprechen zwei Befehlen (zum Beispiel INC und SBC), wieder andere entsprechen dem NOP-Befehl und einige haben so komplizierte Wirkungen, daß sie bei der Programmierung wirklich keinen Nutzen bringen.

Befehl:	D
Syntax:	D (Anfangsadresse) (Endadresse)
Wirkung:	disassembliert Bereich von Anfangsadresse bis Endadresse. Falls die Endadresse fehlt, werden 21 Zeilen von Anfangsadresse an disassembliert, falls kein Parameter übergeben wird: 21 Zeilen von der letzten über M oder D bearbeiteten Adresse an.
englisches Wort:	Disassemble (disassemblieren)

Bild 18. Kurzübersicht zum Befehl D

```
MONITOR
    PC   SR AC XR YR SP
; FB000 00 00 00 00 F8

. F4018  A9 00     LDA #$00
. F401A  85 15     STA $15
. F401C  58        CLI
. F401D  4C 37 4D  JMP $4D37
. F4020  00        BRK
. F4021  FF        ???
. F4022  FF        ???
. F4023  20 7A 41  JSR $417A
```

Bild 19. Beispiel für die Anwendung des D-Befehls

Befehl:	A
Syntax:	A Anfangsadresse Befehl
Wirkung:	assembliert den Befehl ab Anfangsadresse direkt in den Speicher.
englisches Wort:	Assemble (assemblieren)

Bild 20. Kurzübersicht zum Befehl A

Der Assemblier-Befehl: A

Während der Disassemblier-Befehl standardmäßig zu einem Monitor gehört, ist der Assemblier-Befehl erst im Laufe der Zeit zum Standard geworden. Der C128-Monitor kennt einen solchen; er heißt A als Abkürzung für »assemble« (assemblieren). Die Syntax ist (siehe auch Kurzübersicht in Bild 20) »A Anfangsadresse erster Befehl«.

Beispiel: Geben Sie »A 01D00 LDA #$0D« ein. Die Syntax der Mnemonics soll hier nicht weiter beschrieben werden, sie ist die gleiche wie die, die der D-Befehl bei der Ausgabe verwendet.

Wenn eine Zeile nicht korrekt ist, erscheint ein Fragezeichen. Andernfalls – also wenn die Zeile angenommen wurde – erscheint in der nächsten Zeile »A nächste Adresse« und dahinter ein blinkender Cursor als Aufforderung zur Eingabe der nächsten Zeile. Nach unserem Beispiel »A 01D00 LDA #$0D« würde also »A 01D02« erscheinen. Geben Sie dann »JSR $FFD2« ein und drücken Sie <RETURN>. Wieder wird die Zeile angenommen.

Falls Sie aus dem Assemblieren aussteigen wollen, drücken Sie einfach <RETURN>. Nach »JSR $FFD2« sehen Sie dann »A 01D05«. Geben Sie nun »LDA #$21« ein und drücken Sie RETURN, bei der nächsten Zeile geben Sie bitte »JMP $FFD2« ein. Die darauffolgende Zeile wollen wir nicht mehr eingeben. Drücken Sie also einfach ohne Befehlseingabe auf <RETURN>.

Nun können wir unser kleines Maschinenprogramm über »D 01D00« disassemblieren. Wenn wir den Monitor über X verlassen und »BANK15:SYS7424« starten, gibt das Programm zuerst einen Zeilenvorschub (Carriage Return oder CR) und dann ein Ausrufezeichen aus.

Falls Sie nicht wollen, daß CR ausgegeben wird (erste Zeile: LDA #$0D, $0D = #13 = ASCII-Code von CR), müssen Sie unser Programm nur mit »D 1D00« disassemblieren, und in der ersten Zeile den Operand des LDA-Befehls ($0D) mit einem neuen Wert (zum Beispiel $93 für »Bildschirm löschen«) überschreiben.

Durch Drücken von <RETURN> wird eine solche Änderung übernommen. Am Anfang der nächsten Zeile steht dann wieder der A-Befehl, da der Punkt ».« am Beginn einer Disassemblerzeile identisch mit dem A-Befehl ist und somit eine Assemblierung bewirkt.

An dieser Stelle muß noch ein Hinweis für Einsteiger gegeben werden: Wenn der Monitor eine Zeile nicht annimmt, obwohl Sie sicher sind, daß die Zeile syntaktisch korrekt ist,

gibt es mehrere mögliche Ursachen:
- Sie verwenden eine unerlaubte Adressierungsart: JSR ($2000) gibt es beim 8502 ebensowenig wie LDA ($2000),Y (obwohl »LDA ($20),Y« korrekt ist)
- Sie überschreiten bei Branch-Befehlen (BNE, BEQ, BCS und so weiter) die Sprungweite (128 Byte vor und zurück, nicht mehr):
  01D00 BNE F000 ist nicht möglich
  01D00 BNE 1D04 dagegen schon.

Auf jeden Fall ist die Wahrscheinlichkeit, daß am Computer etwas defekt ist, gering, wenn beim A-Befehl etwas nicht funktioniert. Sehen Sie in einem solchen Fall lieber in Ihrer Maschinensprache-Literatur nach.

Sie sehen, das Assemblieren mit dem A-Befehl des TEDMON ist kein Zuckerschlecken, sondern mühsam und fehleranfällig. Vor allem, wenn man an Programmen, die mit diesem »line by line«-Assembler erstellt wurden, Änderungen vornehmen will, stellt sich bald der »Programmierfrust« ein.

Während das Schreiben von Kleinst-Routinen mit dem TEDMON noch gerade möglich ist, wird das Programmieren längerer Routinen so mühselig, daß man (fast) keine Erfolgsaussichten hat. Daher gibt es noch eine höhere Stufe des Assemblerprogrammierens, nämlich die sogenannten Makroassembler. Ein Makroassembler ist ein Programm, das die Eingabe von Maschinenprogrammen wie in einer Textverarbeitung ermöglicht, wobei die Eingaben nicht sofort in den Speicher übernommen werden. Außerdem kann man einzelne Stellen im Programm mit einem Label markieren, kleinere Befehlsfolgen als Makros definieren und vieles mehr.

Mit einem Makroassembler kann man effizient in Assembler programmieren, und auch längere Routinen sind kein Problem mehr. Leider sind solche komfortablen Assembler oft recht teuer (mehrere hundert Mark!). Eine erfreuliche Ausnahme ist da der TOP-ASS von Markt&Technik, der für nur 89 Mark erstaunliche Leistungsmerkmale bietet und es mit jedem 300-Mark-Assembler aufnehmen kann.

Der Sprung-Befehl des Monitors: G

Unser Beispielprogramm (Ausgabe von CR und »!«) aus dem letzten Abschnitt haben wir, um es zu testen, über »BANK Bankeinstellung: SYS Startadresse« gestartet. Da man mit dem Monitor des öfteren in die Lage kommt, daß man ein Maschinenprogramm starten will, gibt es auch hierfür einen Befehl.

Er heißt G (von englisch »to go« = deutsch »gehe«) und hat zwei mögliche Syntaxformen. Die erste Möglichkeit ist »G Adresse« (zum Beispiel »G F1D00« bei unserem Beispielprogramm aus dem letzten Abschnitt). Die zweite ergibt sich daraus, daß man nur »G« (also ohne Adresse oder sonstige Parameter) angibt. Dann wird die Adresse, die als aktueller PC gilt (siehe Abschnitt »TEDMON und Maschinensprache«), als G-Adresse eingesetzt.

In jedem Fall bewirkt der G-Befehl einen Sprung zur jeweiligen Zieladresse. Vorher werden noch die Register mit den Werten, die Sie beim Start des Monitors hatten, geladen. Eventuelle Änderungen der Registerinhalte machen sich erst jetzt bemerkbar!

Befehl:	G
Syntax:	G (Startadresse)
Wirkung:	springt unter Berücksichtigung der Register zur angegebenen Startadresse. Wird diese weggelassen, so zum bei PC angezeigten Wert.
englisches Wort:	Go (gehe)

Bild 21. Kurzübersicht zum Befehl G

Daß die Syntax des G-Befehls eine der einfachsten des Monitors ist, können Sie Bild 21 entnehmen.

Ein Hinweis: Wenn am Ende des angesprungenen Maschinenprogramms ein RTS-Befehl steht, so bewirkt dieser dann keinen Rücksprung in den Monitor; ein solcher wird aber dafür durch den BRK-Befehl bewirkt. Wenn Sie unser Beispielprogramm aus dem letzten Abschnitt mittels »G F1D00« starten, so arbeitet es zwar korrekt; da es aber durch einen RTS-Befehl am Ende der Unterroutine $FFD2 beendet wird, erscheint ein »?SYNTAX ERROR«. Dieser ist jedoch nicht weiter tragisch, da er ja nicht auf einen Programmfehler hinweist.

Eine recht nützliche Anwendung des G-Befehls ist »G FFF3D«. Mit diesem Befehl kann man jederzeit aus dem Monitor heraus einen Reset hervorrufen.

Das Anzeigen der Registerinhalte: R

Wie wir schon erwähnt haben, erfolgt beim Start des Monitors die automatische Registeranzeige. Da man die Register jedoch öfters sehen will – auch längere Zeit nach dem Start des Monitors –, ist ein Befehl nötig, der jederzeit eine Registeranzeige ermöglicht. Dieser heißt einfach R (»register display« = Registeranzeige, siehe auch Bild 22). R arbeitet wie die schon beschriebene Registeranzeige; genauer gesagt, beim Monitorstart wird die Routine zum R-Befehl aufgerufen.

Eine nähere Beschreibung erübrigt sich also. Auch hier gilt, daß man die Register ändern kann.

Die angezeigten oder geänderten Registerinhalte werden beim G-Befehl, den wir im letzten Abschnitt behandelt haben, endgültig in die tatsächlichen Prozessorregister übernommen.

Um etwas Übung mit den Befehlen R (und »;« zum Ändern der Registerinhalte) und G zu bekommen, wollen wir eine kleine Anwendung durchgehen.

Wenn man die Einsprungadresse $FFD2 in Bank $F (Bank 15) anspringt – wie es unser Beispiel beim A-Befehl getan hat – wird das Zeichen ausgegeben, dessen ASCII-Code im Akkumulator steht.

Wir wollen nun den CHR$-Code $07 (= #7, entspricht dem akustischen Klingelzeichen) ausgeben lassen. Dazu gehen wir folgendermaßen vor:
- R eingeben. Nun erscheinen die Registerinhalte.
- Wert unter »AC« auf 07 ändern.
- »G FFFD2« eingeben. Das Zeichen wird nun ausgegeben. Da die ROM-Routine ab $FFD2 durch einen RTS-Befehl beendet wird, erscheint – wie Sie im letzten Abschnitt über den G-Befehl erfahren haben – ein SYNTAX ERROR. Dieser ist jedoch zu ignorieren.

Probieren Sie doch einmal verschiedene Zeichen durch (zum Beispiel die Steuerzeichen für die Cursor-Farbe oder ähnliche).

Durch dieses Arbeiten mit den Registeranzeigen und Sprungbefehlen ist es möglich, auch eigene Routinen auf Funktionsweise bei bestimmten Registerinhalten zu überprüfen.

Jetzt haben wir alle Befehle

Mittlerweile sind alle Befehle des Monitors besprochen, und Sie werden zugeben, daß das Lernen dieser Befehle wirklich keine Überforderung war (denken Sie einmal an Ihre Basic- oder Maschinensprache-Anfängerzeiten zurück).

Von nun an werden wir die Anwendung des gesamten Monitors, also nicht nur einzelner Befehle, besprechen. Dabei soll uns zuerst die Anwendung auf Maschinenprogramme interessieren, danach werden wir mit der Basic-Anwendung fortfahren.

Die Kooperation TEDMON/Maschinensprache soll vor allem deswegen zuerst behandelt werden, weil die letzten Abschnitte gerade erst die Befehle zur Anwendung auf Maschinenprogramme erläutert haben. Als Nur-Basic-Programmierer, den an Maschinenprogrammen höchstens die ASCII-Tabellen zum Ändern und Eintragen eigener Meldungen interessieren, müssen Sie sich ein wenig gedulden, aber die Anwendungen, die dann folgen, werden Sie mit Sicherheit entschädigen.

Um Ihnen die Arbeit mit den gelernten Befehlen zu erleichtern, finden Sie in Form von Bild 23 eine Übersicht (Referenz) aller TEDMON-Befehle. Damit diese möglichst wenig Platz beansprucht (was das Finden bestimmter Informationen erleichtert), wurde eine möglichst knappe – aber präzise – Form der Syntax-Angabe gewählt.

Der erste Buchstabe einer Befehlszeile ist in jedem Fall das Kommando (zum Beispiel M). Die Parameter werden mit Buchstaben angegeben. Vier Buchstaben (zum Beispiel XXXX) bezeichnen eine vierstellige, zwei Buchstaben eine zweistellige Hexadezimalzahl. Natürlich kann man beim TEDMON erfreulicherweise die Parameter auch in anderen Zahlensystemen angeben. Mit Hilfe der Anzahl der Hex-Stellen ist aber eindeutig festgelegt, in welchem Bereich der Parameter liegen muß:

zwei Stellen = $00 bis $FF = #0 bis #255 = %00000000 bis %11111111
vier Stellen = $0000 bis $FFFF = #0 bis #65535

Das B vor den Zahlen (zum Beispiel BXXXX) zeigt an, daß man im Falle der hexadezimalen Zahlendarstellung zusätzlich die Speicherbank angeben kann.

Eingeklammerte Parameter können stehen, müssen aber nicht. In Anführungszeichen stehende Parameter (nur bei L, S, V) sind Strings und müssen auch beim Monitor in Anführungszeichen stehen.

Suchen nach Maschinensprachebefehlen mittels H

Da jeder Maschinensprachebefehl im Speicher als Opcode (Opcode = OPeration CODE = Verarbeitungscode) abgelegt wird, kann man nach bestimmten Befehlen suchen, indem man den Opcode suchen läßt.

Wenn man aber einen Assembler zur Verfügung hat, der im Editor eine Suchfunktion bietet, so sollte man lieber nach der mnemonischen Schreibweise im Quelltext suchen (sofern man den Quelltext hat und es sich nicht um ein fremdes Programm handelt), da das Suchen nach Opcodes einen großen Nachteil hat: Ein Befehl hat so viele unterschiedliche Opcodes wie Adressierungsarten.

Dieser Nachteil ist selbstverständlich belanglos, wenn man die Adressierungsart, die man suchen will, kennt, oder der Befehl nur eine einzige Adressierungsart hat (was nur bei impliziten Befehlen wie SEI, CLD, INX und so weiter der Fall ist). Das Suchen nach einem unmittelbar-adressierten LDA-Befehl (in mnemonischer Schreibweise) erfordert nur die Suche nach einem einzigen Opcode (nämlich $A9, dem Opcode für LDA unmittelbar), wogegen man bei der Suche nach allen LDA-Befehlen (also unabhängig von der Adressierungsart) nach acht Opcodes suchen muß:

Befehl:	R
Syntax:	R (keine Parameter)
Wirkung:	zeigt Register und Registerinhalte an.
englisches Wort:	Register display (Registeranzeige)

Bild 22. Kurzübersicht zum Befehl R

Befehl	Kurzbeschreibung	Syntax
A	Assemblieren	A BXXXX Befehl
C	Vergleich	C BXXXX BYYYY BZZZZ
D	Disassemblieren	D (BXXXX) (BYYYY)
F	Füllen	F BXXXX BYYYY ZZ
G	Programm starten	G (BXXXX)
H	Suchen	H BXXXX BYYYY Z1 Z2 ... ZZ oder H BXXXX BYYYY "TEXT
L	Laden	L ("NAME")(,XX),(BYYY)
M	Memory Dump	M (BXXXX) (BYYYY)
R	Registeranzeige	R
S	Speichern	S "NAME",XX,BYYY BZZZZ
T	Verschieben	T BXXXX BYYYY BZZZZ
V	Verifizieren	V "NAME"(,XX)(BYYYY)
X	Verlassen	X
>	Ändern	>BXXXX Y1 Y2 ... YY
@	Floppy-Bedienung	<Kl.–a.>(XX)(,BEFEHL)

Bild 23. Alphabetische Übersicht über alle Befehle des TEDMON

Opcode Adressierungsart mnemonische Schreibweise (Bsp)
$AD absolut LDA $4567
$A5 Zeropage LDA $45
$A9 unmittelbar LDA #$45
$BD absolut-x-indiziert LDA $4567,X
$B9 absolut-y-indiziert LDA $4567,Y
$A1 indirekt-x-indiziert LDA ($45,X)
$B1 indirekt-y-indiziert LDA ($45),Y
$B5 Zeropage,X-indiziert LDA $45,X

Das heißt, daß Sie bei der Suche nach allen LDA-Befehlen im Bereich $4000 bis $5000 in Bank 0 folgende Befehle benötigen würden:

H 4000 5000 AD sucht die absoluten LDAs,
H 4000 5000 A5 sucht die Zeropage-adressierten,
H 4000 5000 A9 sucht die unmittelbar-adressierten,
H 4000 5000 BD sucht die absolut-x-indizierten,
H 4000 5000 B9 sucht die absolut-y-indizierten,
H 4000 5000 A1 sucht die indirekt-x-indizierten,
H 4000 5000 B1 sucht die indirekt-y-indizierten,
H 4000 5000 B5 sucht die Zeropage-x-indizierten LDAs.

Sie werden – falls im angegebenen Suchbereich ein Maschinenprogramm und nicht zufällige Daten aufbewahrt sind – eine Unmenge Adressen erhalten. Es wäre allerdings ein Trugschluß, daß an jeder gefundenen Adresse wirklich ein solcher Befehl steht (»stehen« muß zwar einer, wenn auch der Opcode vorhanden ist, aber die Frage ist, ob dieser innerhalb eines Maschinenprogramms oder innerhalb von Daten steht und womöglich nie durchlaufen wird).

Es empfiehlt sich daher, jede angegebene Adresse genauer zu prüfen; da es sich unter Umständen um recht viele handelt, ist die Ausgabe der gefundenen Adressen auf den Drucker (siehe Abschnitt über Ein-/Ausgabe) wirklich empfehlenswert.

Zur Prüfung einer bestimmten Adresse darauf, ob es sich wirklich um einen Befehl handelt, sollte man einfach »D Adresse« eingeben. Bei der angegebenen Adresse steht dann der aufgespürte Befehl; da wir nach den Opcodes gesucht haben, muß dies ja auch der Fall sein.

Um nun festzustellen, ob dieser Befehl inmitten eines Teils eines Maschinenprogramms steht, sind die dem Befehl unmittelbar folgenden Anweisungen zu überprüfen. Handelt es sich dabei hauptsächlich um keine Befehle (dann erscheinen ja an den betreffenden Adressen Fragezeichen im Disassembler-Listing), so gibt es nur zwei Möglichkeiten:

1. Es wurden bei der Programmierung undefinierte Opcodes verwendet; diese Möglichkeit ist allerdings die seltenste

und nur bei kopiergeschützten Programmen denkbar.
2. Es handelt sich schlicht um keinen Befehl, sondern nur um Daten. Ein Unterfall dieser Möglichkeit ist, daß der Befehlscode im Operand eines anderen Befehls vorhanden ist (im Speicherformat von LDA #$2C - $A9 $2C - ist zum Beispiel mit $2C der Opcode des BIT-absolut-Befehls vorhanden).

Der zweite Fall ist - wie gesagt - der eindeutig häufigere. In jedem dieser Fälle kann man mit 95prozentiger Wahrscheinlichkeit davon ausgehen, daß es sich um keinen echten Befehl handelt. Falls natürlich ordentliche Befehle erscheinen, so kann man gar mit 99prozentiger Sicherheit annehmen, daß es sich um einen »echten« Befehl handelt.

In diesem Zusammenhang möchten wir Ihnen ein interessantes Verfahren zur Suche nach einem Bereich, in dem Maschinenprogramme liegen, vorstellen. Wenn man zum Beispiel ein fremdes Programm hat, in dem viele Tabellen (geordnete Daten) untergebracht sind, möchte man vielleicht wissen, wo denn eigentlich das Programm liegt.

Natürlich kann man das ganze Programm disassemblieren lassen und dann einfach suchen, aber zur Grobeingrenzung des Bereichs ist es ungleich komfortabler, den H-Befehl einzusetzen.

In »normalen« Maschinenprogrammen ist so ziemlich der häufigste Opcode, der nicht als ASCII- oder Bildschirm-Code eine sinnvolle Bedeutung hat wie etwa $20 (Opcode für JSR, ASCII-Code für Space, Bildschirmcode für Space), der Opcode für »LDA #«, also $A9.

Mit »H Anfangsadresse Endadresse A9« kann man nun den gesamten Bereich durchsuchen lassen. An den Stellen, an denen das Suchbyte $A9 gedrängt auftritt, kann man annehmen, daß es sich um einen Bereich handelt, der vorwiegend aus Maschinensprachebefehlen besteht.

Wenn man die Bedeutung einzelner Adressen in einem bestimmten Programm - zum Beispiel dem Betriebssystem bei der Erstellung eines kommentierten ROM-Listings - näher durchleuchten will, kann natürlich auch hier der H-Befehl, von dem Sie jetzt nach und nach merken, wie vielseitig er ist (auch wenn es auf den ersten Blick gar nicht so aussieht), eingesetzt werden.

Wir müssen alle Befehle suchen, die auf die Adresse zugreifen. Bei Zeropage-Adressen geht man folgendermaßen vor (nehmen wir das Beispiel der Adresse $15):

1. »H Anfangsadresse Endadresse Adresse«,
also zum Beispiel »H F4000 F4200 15«

Sie erhalten dann alle Adressen, an denen $15 steht. Wenn Sie jedoch »D gefundene Adresse« eingeben, werden Sie keinen Befehl, der auf $15 zugreift, finden.

Die Adresse, ab der dann überhaupt ein Befehl stehen kann, der die Zeropage-Adresse als Operand hat, ist die gefundene Adresse minus 1.

2. Test der gefundenen Adresse

Das Testen, ob die gefundenen Werte wirklich Rückschlüsse auf einen Befehl, der die gesuchte Zeropage-Adresse bearbeitet, zulassen, geht relativ einfach.

Sie müssen nur »D Adresse-1« eingeben, und schon sehen Sie den entsprechenden Befehl disassembliert. Wenn es sich um einen Befehl, der auf die gesuchte Adresse zugreift, handelt, so sehen Sie ein Disassembler-Listing von diesem. Andernfalls steht ab der gefundenen Adresse minus 1 kein Befehl (im Disassembler-Listing sehen Sie Fragezeichen) oder ein Branch-Befehl (zum Beispiel ein BEQ, BCS- oder anderer Befehl, der zwar als Parameter die Zeropage-Adresse hat, dieser Parameter aber die Sprungweite anstelle einer Adreßangabe ist) oder ein Befehl, der auf eine Adresse zugreift, die zwar als LB den gleichen Wert wie die Zeropage-Adresse hat, aber ein HB ungleich 0.

Ein Beispiel: Wenn Sie ab den gefundenen Adressen-1 disassemblieren und nach Zugriffen auf $15 suchen, sind beispielsweise folgende Befehle richtig:

LDA $15
LDA $0015 (ist (fast) das gleiche wie LDA $15)
STA $15
JMP $0015
BIT $15
ADC $15

Die folgenden Befehle greifen nicht auf $15 zu:

LDA $4015 (HB ungleich 0)
LDA #$15 (unmittelbar-adressiert)
BNE xxxx ($15 war nur eine Angabe der Sprungweite)
BIT $3015 (HB ungleich 0)
CLD (CLD hat keinen Parameter, $15 wäre dann ein Befehlscode)

Mit der Zeit bekommt man schon Gespür dafür, ob es sich um einen Zugriff auf eine Adresse handelt oder nicht.

3. Prüfung, ob der Befehl durchlaufen wird

Nun bleibt noch abzuwarten, ob der Befehl jemals abgearbeitet wird oder nicht. Wie man dies feststellt, haben wir schon bei der Suche nach den Opcodes der Befehle geklärt.

Bei der Arbeit an Programmen, die illegale Opcodes in sich haben, ist natürlich jegliche Entscheidung erschwert.

Als nächster Punkt soll die Suche nach Befehlen, die auf beliebige Zwei-Byte-Adressen ($0100 bis $FFFF) zugreifen, besprochen werden.

Suchen nach Zugriffen auf beliebige Adressen

Da wir schon wissen, wie man Zugriffe auf Zeropage-Adressen aufspürt, soll uns nun interessieren, wie man solche auf Adressen, die mit zwei Byte (LB und HB) darstellbar sind, sucht.

Außer des zusätzlichen (zu suchenden) Bytes der Adreßangabe (statt ein Byte nun zwei) ändert sich nichts Grundlegendes.

Während man bei der hexadezimalen Schreibweise zuerst das HB und dann erst das LB nennt ($4567; $45 = HB, $67 = LB), legt der Computer diese Daten im Speicher genau umgekehrt ab (diese Darstellung ist für die CPU günstiger, der Grund dafür würde uns aber viel zu weit vom Thema abbringen und soll daher vernachlässigt werden):

JMP $4567 wird abgelegt als $4C $67 $45

Nun haben wir unser theoretisches Wissen vervollständigt und müssen es nur noch in die Praxis umsetzen. Dies ist das Verfahren zur Suche der Befehle, die auf eine mit zwei Byte (Low-Byte=LB, High-Byte=HB) dargestellte Adresse zugreifen:

1. »H Anfangsadresse Endadresse LB HB«,
im Beispiel bei der Suche nach Zugriffen auf $4567:
»H Anfangsadresse Endadresse 67 45«
2. Test der gefundenen Adresse

Ob ab der gefundenen Adresse minus 1 (1 abziehen, da Adresse des LB im Speicher genannt wird, wir aber die Adresse des ein Byte vorher im Speicher stehenden Befehlscodes benötigen) ein »richtiger« Befehl steht, wird wie gehabt überprüft (»D Adresse-1« und so weiter, siehe »Vorgehensweise bei Zeropage-Adressen«).

3. Prüfung, ob Befehl durchlaufen wird
Auch das haben wir bereits geklärt.

An dieser Stelle ist ein wichtiger Hinweis zu machen, der vor allem den Einsteigern unter Ihnen zugedacht ist.

Wenn Sie die Verzweigungs(englisch »branch«)-Befehle, die eine bestimmte Adresse anspringen, suchen, so können Sie mit den geschilderten Verfahren die entsprechenden Branches nicht ausfindig machen. Der Grund ist, daß ein Bxx-(BEQ, BCS, ...)-Befehl folgendermaßen im Speicher abgelegt wird:

Befehlscode Entfernung zum Sprungziel

Die Entfernung zum Sprungziel wird durch eine vorzeichenbehaftete Binärzahl dargestellt.

Sie können aus dem obigen Schema ersehen, daß das Sprungziel nicht direkt (wie etwa bei JMP, JSR, LDA, STA, INC, ...) im Speicher abgelegt ist (sondern nur die relative Sprungweite dahin) und somit ein H-Befehl nach LB und HB erfolglos bleibt.

Wenn man nun die Verzweigungsbefehle über H suchen wollte, so müßte man für jede untersuchte Adresse die Sprungweite neu berechnen, und dies ist wirklich zu mühsam. Es empfiehlt sich daher, den in Frage kommenden Bereich zu disassemblieren und durch eigene Suche die entsprechenden Befehle ausfindig zu machen. Dies ist allerdings aufgrund einer wichtigen Tatsache nicht so schlimm:

Verzweigungsbefehle haben eine stark begrenzte Sprungweite!

Die Sprungweite reicht nur 128 Byte nach hinten und 127 nach vorn, so daß sich ein gerade 256 Byte großer Bereich ergibt, den man mit »D« betrachten muß.

Suchen nach kompletten Befehlen

Nun soll es für uns kein Problem mehr sein, auch nach ganzen Befehlen (zum Beispiel LDA $1245 oder JSR $FFD2) suchen zu lassen. Dazu brauchen wir nur zwei Schritte:

1. Suche über H
2. Test, ob Befehl durchlaufen wird

Wie man den zweiten Punkt durcharbeitet, haben wir bereits besprochen. Nun interessiert uns naturgemäß der erste Punkt, aber auch dieser ist nicht völlig neu und wirklich keine große Schwierigkeit.

Dazu wollen wir den bequemsten Weg wählen, denn die Umrechnung der mnemonischen Schreibweise in den Code ist zwar nötig (weil der H-Befehl nur nach Zahlen und nicht nach anderen Daten suchen kann), aber der A-Befehl kann uns dabei in hervorragender Weise unterstützen.

Wir müssen nämlich nur »A 400 umzurechnender Befehl« eingeben und sofort den Assemblier-Modus verlassen (über <RETURN>). In der Zeile, in der wir den Befehl assembliert haben, sehen wir dann nach der Adresse (aber vor der mnemonischen Schreibweise) die hexadezimale Darstellung des Speicherformats.

Wir haben dazu einfach in den 40-Zeichen-Bildschirmspeicher (ab $400) assembliert; dies ist bei Verwendung des 80-Zeichen-Bildschirms unproblematisch (sofern nicht bei $400 ein Maschinenprogramm liegt), bei der Arbeit mit 40 Zeichen pro Zeile sollte man folgendermaßen vorgehen, damit wirklich nichts schiefgehen kann:

- Bildschirm mit <SHIFT+CLR/HOME> löschen
- einmal <RETURN> drücken

Beim Assemblieren wird der erzeugte Code des Befehls (ein bis drei Byte) ab $400 abgelegt; deshalb erscheinen links oben am Bildschirm Zeichen. Damit diese Zeichen nicht ihrerseits durch die Neu-Anzeige nach dem Assemblieren gelöscht werden, ist es günstiger, die Eingabe nicht in der obersten Bildschirmzeile zu machen.

Nun wollen wir als Beispiel nach dem Befehl »JSR $FFD2« im Bereich $4000 bis $4500 in Bank $F (#15) suchen.

Der erste Schritt ist die Umwandlung des Befehls ins Speicherformat. Dazu geben wir folgende Zeile ein:

A 400 JSR $FFD2

Diese Eingabezeile wird sogleich vom Monitor überschrieben durch

A 00400 20 D2 FF JSR $FFD2
A 00403 (dahinter Cursor als Eingabe-Aufforderung)

Nun drücken wir <RETURN>, um den Assemblier-Modus zu verlassen. Wir wissen nun, daß der Programmcode zu JSR $FFD2 folgendermaßen 20 D2 FF lautet.

Nach diesen Codes können wir nun im Bereich $4000 bis $4500, den wir ja durchforsten wollen, suchen lassen:
H F4000 F4500 20 D2 FF

Um die gefundenen Befehle zu sehen (zum Beispiel um sie zu testen), brauchen wir jetzt keine Zahl subtrahieren, da die angegebenen Adressen schon stimmen; da wir auch den Befehlscode ($20 für JSR) suchen lassen, wird immer die Adresse angegeben, ab der $20 (der Befehlscode) steht.

Dieser Abschnitt hat Ihnen hoffentlich gezeigt, wieviel man aus dem H-Befehl machen kann, wenn man nur das nötige Know-how hat - und das wird Ihnen hier nach und nach vermittelt.

Analyse eines Maschinenprogramms mit TEDMON

Maschinensprache lernt man auch dadurch, daß man fremde Maschinenprogramme analysiert und sich so bestimmte Programmstrukturen aneignet.

Hierbei ist der Monitor eine unentbehrliche Hilfe, wenn es darum geht, ein Programm zu durchleuchten. Dies kann man selbstredend nicht nur zu Lehrzwecken tun, sondern auch, um Änderungen an einem Programm durchzuführen oder es gar zu knacken (wenn man sich ganz gut auskennt). Auch die großen »Knacker« - die »Produzenten« unzähliger Raubkopien - arbeiten mit Monitoren!

Wir möchten Ihnen hier ein schrittweises Vorgehen, das Sie an einem Beispielprogramm Ihrer Wahl (es sollte nicht allzu lang sein) nachvollziehen können, vorstellen. Erforderlich ist dazu zwar ein Drucker, aber für ernsthafte Anwendungen wie Maschinensprache ist ein solcher ohnehin erforderlich.

Einige Schritte unseres Verfahrens haben wir schon angesprochen oder gar ausführlich behandelt, so daß das Verfahren wirklich von jedem Maschinenprogrammierer, der dies liest, angewendet werden kann.

Schritt 1: Bestimmung, wo Befehle und wo Tabellen stehen

Als erstes sollte man sich vom Maschinenprogramm eine genaue Liste zusammenstellen, wo Maschinenroutinen stehen (also ausführbare Befehle, die auch durchlaufen werden) und wo nur Daten (zum Beispiel ASCII-Tabellen für Textausgabe) aufbewahrt sind.

Eine solche Liste würde für den ROM-Bereich $4000 bis $4297 (natürlich in Bank 15 = $F, da es sich um ROM handelt) etwa folgendermaßen aussehen:

4000 bis 401F: Programmcode (Befehle)
4020 bis 4022: undefinierte Bytes (»Abfall«)
4023 bis 41BA: Programmcode
41BB bis 4250: ASCII-Tabelle (Einschaltmeldung)
4251 bis 4266: Programmcode
4267 bis 4278: numerische Tabelle
4279 bis 4297: Programmcode

Wenn wir nun das Programm ausgeben wollen, wird der Programmcode disassembliert (mittels D), während die Daten mittels M gedumpt werden müssen (ein Disassembler-Listing würde kein sinnvolles Ergebnis bringen).

Bei Programmcode könnte man übrigens noch prüfen, ob es sich um eine Unterroutine handelt; Unterroutinen erkennt man daran, daß ein RTS-Befehl am Ende steht. Falls ja, so sollte man hinter »Programmcode« in der Liste noch den Vermerk »UP« (UnterProgramm) anbringen, denn dies kann hinterher die Arbeit stark erleichtern.

In unserem Beispiel - einem Teil des C128-ROMs - könnte man diesen Vermerk zum Beispiel bei den Adressen 4251 bis 4266 oder 4279 bis 4297 anbringen.

Mit diesem ersten Schritt, der durchaus zeitaufwendig sein kann, haben wir schon einen großen Teil geleistet.

Schritt 2: Ausgabe des Listings auf dem Drucker



Dieser zweite Schritt ist keine Herausforderung mehr, wenn man den ersten Schritt sorgfältig erledigt hat. Im Beispiel würden wir den ROM-Ausschnitt folgendermaßen ausgeben lassen:
OPEN4,4:CMD4:MONITOR (Druckerausgabe)
D F4000 F401F (Programmcode disassemblieren)
M F4020 F4022 (Daten dumpen)
D F4023 F41BA (Programmcode disassemblieren)
M F41BB F4250 (Daten dumpen)
D F4251 F4266 (Programmcode disassemblieren)
M F4267 F4278 (Daten dumpen)
D F4279 F4297 (Programmcode disassemblieren)
X (Monitor verlassen)
PRINT#4:CLOSE4 (Ein-/Ausgabe initialisieren)

Natürlich kann man die Ausgabe auf den Bildschirm umleiten, aber dies ist nicht sinnvoll, da der Bildschirm nur wenige Zeilen Listing faßt.

Außerdem ist ein Listing auf dem Papier viel leichter zu bearbeiten, da die Daten nicht nach dem Ausschalten des Computers verloren sind und man nicht dauernd auf den Monitor oder Fernseher sehen muß.

Schritt 3: Erste Vermerke im Listing anbringen

Wenn man das Listing (jetzt ist es ja auf dem Druckerpapier) übersichtlicher machen will, so sollte man vermerken, wo eine Routine endet. Dies ist ganz einfach aus den Maschinenbefehlen ersichtlich; nach einem JMP wird das Programm an einer ganz anderen Stelle fortgesetzt, nach einem BRK abgebrochen, nach einem RTS zum aufrufenden Programm zurückgesprungen oder nach einem RTI die Interrupt-Routine beendet.

Nach folgenden Befehlen sollte man also mit Lineal und Kugelschreiber eine Linie als Trennungsmarke ziehen:
JMP, RTS, BRK, RTI

Unter JSR gehört natürlich keine Linie, da das Programm durch JSR in der Regel nur temporär an anderer Stelle fortgesetzt wird.

Es mag vielleicht eine stumpfsinnige Arbeit sein, aber es lohnt sich wirklich, wenn man auf einen Blick sieht, wo ein Teil des Programms endet!

Nach allen diesen Schritten haben wir schon ein beachtliches Ergebnis vor uns: ein übersichtliches Programmlisting auf Druckerpapier, das nur noch mit Kommentaren versehen werden muß – dieses Kommentieren ist zwar nicht ganz so einfach, aber mit ein wenig Systematik geht es schon viel leichter!

Schritt 4: Hervorhebung von Verzweigungen mit Pfeilen

Eine dem letzten Schritt ähnliche Arbeitserleichterung ist das Verdeutlichen von Verzweigungsbefehlen (BEQ, BNE, BCS, BCC, BVC,..) mit Hilfe von Pfeilen.

Ein Pfeil dürfte für einen Sprung das deutlichste Symbol sein, das auf einen Blick alles aussagt.

Am besten bringen Sie den Pfeil so an, daß er links von der mnemonischen Darstellung des Verzweigungsbefehls beginnt und die Spitze auf die mnemonische Darstellung des Befehls, der am Sprungziel steht und im Falle einer erfüllten Verzweigungsbedingung angesprungen wird. Dann ist sofort ersichtlich, was ein Verzweigungsbefehl bewirkt.

Natürlich ist diese Pfeilmarkierung nicht bei allen Befehlen nötig und bei sehr weiten Verzweigungen auch nicht mehr möglich, aber viele Programmierer haben sicher schon die Erfahrung gemacht, daß ein mit Pfeilen versehenes Listing sehr viel einleuchtender ist.

Schritt 5: Aufrufe von ROM-Routinen kommentieren

Auch den (für diese Zwecke glücklichen) Umstand, daß fast kein Maschinenprogramm ohne ROM-Routine auskommt, können wir uns bei der Analyse eines fremden Maschinenprogramms zunutze machen.

Wir benötigen nur genügend Informationen über die Einsprungadressen und ihre Bedeutung, und wenn wir einen JMP- oder JSR-Befehl ins ROM finden, so können wir hinter diesen Befehl einfach die Bedeutung der ROM-Routine schreiben. Oft stehen vor dem Befehl noch andere Befehle, die für die Parameterübergabe an die ROM-Routine zuständig sind, und auch diese Befehle können wir nun entschleiern.

In der Regel kann man mit der Erklärung von ROM-Aufrufen einen großen Teil des Maschinenprogramms durchleuchten, so daß die restliche Analyse unverhältnismäßig stark erleichtert ist.

Grundvoraussetzung für diesen fünften Schritt ist, daß man eine Tabelle mit ROM-Einsprüngen zur Verfügung hat, um dann die ROM-Routinen-Aufrufe deuten zu können. Eine solche Tabelle braucht ohnehin jeder Maschinenprogrammierer.

Schritt 6: Zusammengehörige Programmteile kennzeichnen

Nun wollen wir noch feststellen, wo mehrere Befehle zusammen ein Modul ergeben, worunter wir Schleifen oder ähnliche Programmteile verstehen.

Schleifen sind auch an den schon in Schritt vier angebrachten Verzweigungspfeilen leicht erkennbar, und solche zusammengehörige Programmteile wie etwa eine Schleife kennzeichnet man am besten mit einer geschweiften (oder auch eckigen, wenn Sie wollen...) Klammer, damit man hinterher genau sieht, wo ein paar Befehle zusammengehören.

Die geschweifte Klammer paßt am besten unmittelbar hinter die mnemonische Schreibweise der Befehle, vor den Mnemonics wurden ja in Schritt vier die Verzweigungspfeile angebracht.

In unserem Beispiel – einem Teil des C 128-ROMs – könnte man zum Beispiel die Befehle $F406A bis $F4073 als Schleife behandeln (sehen Sie doch einmal mit »D F406A F4073« nach) und hinter die mnemonische Schreibweise eine Klammer setzen. Wenn man dann ausgetüftelt hat, was die Schleife bewirkt, so schreibt man diese Erkenntnis hinter die Klammer. Es ist nämlich wenig sinnvoll, Grundstrukturen wie diese Verschiebeschleife Befehl für Befehl zu durchleuchten; vernünftiger ist es, möglichst viele Befehle mit einer umfassenden Erklärung zu versehen, damit man sich nicht in unwesentlichen Einzelheiten wie den Befehlen, die den Schleifenablauf steuern, verliert.

Schritt 7: Mit G-Befehl bestimmte Einsprünge testen

Wie man mit dem G-Befehl und der Möglichkeit des Register-Änderns die einzelnen Routinen eines Programms testen kann, haben wir bereits behandelt. Dies ist vor allem bei der Analyse von Maschinenprogrammen sehr nützlich.

Die vorgestellten sieben Schritte sind der Einstieg in die Analyse eines Programms; nun muß man noch nach und nach auswerten und die »Forschungsergebnisse« im Listing eintragen; in der Regel geht dies zwar langsam, aber die Analyse kostet nun einmal Zeit – viel Zeit.

Dafür lernt man daraus unglaublich viel für die eigene Maschinenprogrammierung; zudem wird man bei dieser Analyse immer sicherer und routinierter, so daß diese bald zügig vonstatten geht.

Vergleich verschiedener Versionen von Maschinenprogrammen

Dieser Abschnitt ist für diejenigen besonders interessant, die Maschinenroutinen mit dem A-Befehl des TEDMON erstellen. Daß diese Form der Maschinenprogrammierung viel zu umständlich ist und die Anschaffung eines guten Assemblerpakets mehr als lohnend ist, braucht nicht weiter erwähnt zu werden. Für kleinere Routinen wird man aber doch manchmal den TEDMON einsetzen.

Da man von einem Maschinenprogramm jede Zwischenversion speichert (zumindest sollte man dies tun, wenn man

nicht die Programmierarbeit leichtfertig Gefahren aussetzen will), möchte man auch einmal zwei Versionen vergleichen. Zwar kann man dies von Hand tun – die C 128-Window-Technik mag ganz nützlich beim parallelen Disassemblieren sein –, aber bei längeren Programmen ist ein solches Vorgehen eine Tortur. Es empfiehlt sich also, den C-Befehl des TEDMON einzusetzen (zumal dieser ja nur darauf wartet, uns zu helfen).

Dazu möchten wir Ihnen wieder ein in Schritte eingeteiltes Verfahren vorstellen.

I. aktuelle Version in Originalbereich laden

Da man nach dem Vergleich an der aktuellen Version weiterarbeiten will, soll diese gleich in den Originalbereich kommen. Am besten lädt man über BLOAD "NAME" oder L "NAME", Gerät im Monitor, also jedesmal ohne Angabe einer Ladeadresse. Dann wird nämlich das Programm in den Bereich, aus dem heraus es gespeichert wurde, geladen.

II. Vor-Version zum Vergleich in anderen Bereich laden

Wenn der V-Befehl die Adresse, an der ein Unterschied auftritt, anzeigen würde (leider tut er das nicht), könnten wir jetzt mit V "ALTE VERSION", Gerät, die neue mit der alten Version vergleichen lassen. Da dies wie gesagt nicht möglich ist, müssen wir die Version, mit der wir vergleichen wollen, in einen anderen Bereich laden (über relatives Laden): L "ALTE VERSION", Gerät, freier Bereich.

Als freien Bereich sollte man einen Speicherbereich nehmen, in dem die Vergleichsversion vorübergehend liegen kann.

III. Vergleich mit C durchführen lassen

Nun können wir mit Hilfe des C-Befehls vergleichen lassen:
C Anfang.neu Ende.neu Anfang.Vergleichsversion
Dabei ist:

Anfang.neu:	Anfang des Originalbereichs, in den als Schritt I die neue Version geladen wurde
Ende.neu:	Ende dieses Originalbereichs (siehe oben)
Anfang.Vergleichsversion:	Anfangsadresse des Bereichs, in dem die Vergleichsversion liegt. Das ist der freie Bereich, in den wir die alte Version als Schritt II geladen haben.

Es werden bei Ausführung des C-Befehls dann alle die Adressen angezeigt, an denen sich die neue Version von der alten unterscheidet. Nun können Sie an der neuen Version weiterprogrammieren!

```
100 REM "PROGRAMM VERSUCHSKANINCHEN"
110 SCNCLR:PRINT"(C) 64'ER"
120 REM ****************************
130 COLOR6,1
140 END
```

Listing 7. Das »Versuchskaninchen«

Nun geht es an die Basic-Programme

Ab jetzt sollen uns nur noch die Anwendungen des Monitors auf Basic-Programme interessieren, denn darüber findet man sonst nichts in der Literatur.

Für Maschinenprogrammierer sei gesagt, daß sie die gegebenen Hinweise auch mitverfolgen sollten, da sie sowohl etwas über die Arbeitsweise ihres Computers erfahren als auch die Anwendungsbeispiele für Maschinenprogramme nutzen können: Viele Maschinenprogramme haben an ihrem Anfang eine Basic-Zeile, in der dann der SYS-Befehl steht, der erst das eigentliche Maschinenprogramm startet.

Wie man dafür noch interessante Effekte einsetzen kann, erfahren Sie ganz am Ende des Kurses.

```
MONITOR
    PC  SR AC XR YR SP
; FB000 00 00 00 00 F8

>01C00 00 24 1C 64 00 8F 20 22 50 52 4F 47 52 41 4D 4D:
>01C10 20 56 45 52 53 55 43 48 53 4B 41 4E 49 4E 43 48:
>01C20 45 4E 22 00 37 1C 6E 00 E8 3A 99 22 28 43 29 20:
>01C30 36 34 27 45 52 22 00 5A 1C 78 00 8F 20 2A 2A 2A:
>01C40 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A:
>01C50 2A 2A 2A 2A 2A 2A 2A 2A 2A 00 63 1C 82 00 E7 36:
>01C60 2C 31 00 69 1C 8C 00 80 00 00 00 FF FF FF FF FF:
```

Bild 24. Speicherausdruck des Programms »Versuchskaninchen«

DUMP MIT 8 BYTES PRO ZEILE

```
MONITOR
    PC  SR AC XR YR SP
; FB000 00 00 00 00 F8

>01C00 00 24 1C 64 00 8F 20 22:
>01C08 50 52 4F 47 52 41 4D 4D:
>01C10 20 56 45 52 53 55 43 48:
>01C18 53 4B 41 4E 49 4E 43 48:
>01C20 45 4E 22 00 37 1C 6E 00:
>01C28 E8 3A 99 22 28 43 29 20:
>01C30 36 34 27 45 52 22 00 5A:
>01C38 1C 78 00 8F 20 2A 2A 2A:
>01C40 2A 2A 2A 2A 2A 2A 2A 2A:
>01C48 2A 2A 2A 2A 2A 2A 2A 2A:
>01C50 2A 2A 2A 2A 2A 2A 2A 2A:
>01C58 2A 00 63 1C 82 00 E7 36:
>01C60 2C 31 00 69 1C 8C 00 80:
>01C68 00 00 00 FF FF FF FF FF:
```

Wir werden jetzt kennenlernen, wie ein Basic-Programm im Speicher abgelegt ist (das ist nicht kompliziert, aber sehr interessant, und wenn man es begriffen hat, kann man viel damit anfangen) und wollen dann unser Wissen zu Manipulationen an Basic-Programmen einsetzen. Rund um das Thema »Listschutz« wird alles besprochen werden – und noch ein bißchen mehr –, so daß Sie dann in der Lage sind,

a) eigene Programme zu schützen
b) Ihre Basic-Listings übersichtlich zu gestalten
c) fremde Schutzvorrichtungen zu entfernen
d) und – last but not least – den Monitor als Unterstützung beim Basic-Programmieren einzusetzen

Die dazu erforderliche Theorie wollen wir, damit es etwas leichter geht, mit TEDMON erarbeiten.

Unser Übungsprogramm, an dem wir von jetzt an dauernd üben wollen, finden Sie als Listing 7 abgedruckt – das Programm heißt »Versuchskaninchen«.

Zum Eintippen des Programms ist noch folgendes zu sagen:
- Schalten Sie vorher den Computer aus und wieder ein oder lösen Sie zumindest einen Reset aus
- Tippen Sie das Programm Zeichen für Zeichen genauso ab, wie es abgedruckt ist
- In Zeile 130 sind es genau 28 Sternchen

Das Programm belegt im Basic-Speicher (Bank 0, da in Bank 0 das Programm aufbewahrt wird) die Speicherplätze $1C00 bis $1C6A. Als Bild 24 sehen Sie das Programm »Versuchskaninchen« in 3 verschiedenen Darstellungsformen:
1. Memory-Dump im 80-Zeichen-Modus (16 Byte pro Zeile)
2. Memory-Dump im 40-Zeichen-Modus (8 Byte pro Zeile)
3. als Listing wie über LIST erreichbar

In den folgenden Erklärungen werden wir uns immer auf Punkt 1, das Memory-Dump, beziehen.

Der Aufbau eines Basic-Programms im Speicher

Beginnen wir gleich mit dem ersten Byte, dem Inhalt der Speicherzelle $1C00. Da wir mit »M 2D« zur Feststellung der Anfangsadresse des Basic-Programms im Speicher sehen können, daß das Basic-Programm im Speicher erst 1 Byte später abgelegt ist (nämlich ab $1C01), mag es zunächst ein wenig verwundern, wenn hier ab $1C00 (also Anfangsadresse-1) gedumpt wird.

In dieser Anfangsadresse-1 muß ein $00-Byte enthalten sein, da sonst bei RUN oder NEW ein SYNTAX ERROR erfolgt (ändern Sie doch einmal im Dump das $00 in $40 oder $01 oder irgendein Byte außer $00, und geben Sie nach X den Basic-Befehl RUN ein, dann sehen Sie den Effekt. Genaugenommen hat dieses $00-Byte keine weitere Funktion, als eine Markierung zu sein.

Diese Markierung haben wir schon beim Verschieben eines Basic-Programms mit dem T-Befehl angesprochen, wo wir diese mittels POKE auf 0 gesetzt hatten.

Bevor wir nun fortfahren, sei noch klargestellt, daß der Interpreter bei der Speicherung von Integerwerten zwischen 0 und 65535 das sogenannte »Low-High-Format« verwendet.

Wie in einem Vektor oder Zeiger (wir erinnern uns an $2D/$2E als Zeiger auf den Beginn des Basic-Programms im Speicher) wird in der Speicherzelle, die eine niedrigere Adreßzahl hat, das Low-Byte, in der darauffolgenden das High-Byte abgelegt. Da im Speicher – insbesondere beim Dump – zuerst das Low-, und dann erst das High-Byte steht, spricht man vom »Low-High-Format«.

Wie man mit diesem umgeht, wurde schon besprochen, und dies hier sollte nur eine kleine Auffrischung sein. Wir wollen aber nun mit dem Thema »Aufbau eines Basic-Programms im Speicher« fortfahren.

Die Linkpointer

Bei $1C01 geht dann schon die Speicherung der ersten Programmzeile – im Beispiel ist das Zeile 100 – los.

Die zwei ersten Bytes einer Zeile sind der »Linkpointer«. »Linkpointer« kommt vom englischen »to link«, was übersetzt »verbinden« heißt.

Die Linkpointer (so sagt man allgemein, und wir wollen dieses Wort beibehalten) zeigen die Adresse an, ab der die nächste Basic-Zeile im Speicher beginnt – da bei der Speicherung am Anfang jeder Zeile der Linkpointer steht, zeigt ein Linkpointer auf den nächsten und so weiter.

Der erste Linkpointer ist also $1C24, bei $1C24 steht ein Linkpointer auf $1C37, dort steht einer auf... Dies geht weiter bis $1C63, wo ein Linkpointer auf $1C69 zeigt. Und bei $1C69? Dort stehen zwei Null-Byte hintereinander als Linkpointer, woran der Basic-Interpreter erkennt, daß an dieser Stelle das Basic-Programm zu Ende ist. Die in dieser Dump-Zeile noch stehenden $FF-Bytes gehören schon nicht mehr zum Programm, und höchstwahrscheinlich stehen bei Ihnen dort andere Werte – zum Beispiel noch Überreste eines vorher im Speicher befindlichen Programms.

Die Hauptaufgabe der Linkpointer ist es, der LIST-Routine die Arbeit zu erleichtern.

Die Zeilennummern

Irgendwo muß sich der Computer auch die Zeilennummer merken. Und diese legt er – ebenfalls im Low-High-Format – unmittelbar hinter dem Linkpointer ab. Im Beispielprogramm stellen die Bytes »64 00« die Zeilennummer der ersten Basic-Zeile dar; wenn wir $0064=$64 umrechnen lassen, sehen wir, daß es sich um Zeile 100 handelt – auch im Listing ist 100 die Nummer der ersten Zeile.

Linkpointer und Zeilennummer sind der »Kopf« einer Basic-Zeile im Speicher. Darauf folgt dann das Speicherformat der Befehle, das uns auch noch interessieren wird.

Die Token

An früherer Stelle wurden die »Token« schon einmal angesprochen. Jetzt wollen wir endlich wissen, worum es sich dabei handelt.

Wenn Sie die ASCII-Darstellung im Dump einmal ansehen, finden Sie zwar den Text »PROGRAMM VERSUCHSKANINCHEN«, der in Zeile 100 auf den REM-Befehl folgt, wieder, aber nicht den REM-Befehl selbst.

Wir wollen nun herausfinden, woher der Computer das REM nimmt. Dazu suchen wir ein Zeichen aus Zeile 100 nach dem anderen. Den Kopf der Zeile (Linkpointer und Zeilennummer) haben wir uns schon angesehen. Auf die Zeilennummer folgt im Listing ein Leerzeichen. Das der Zeilennummer folgende Byte im Speicher ist aber $8F, und nicht $20 (ASCII-Code für Leerzeichen). Dies läßt sich ganz einfach erklären: Ohne besondere Tricks oder Hilfsprogramme werden alle Leerzeichen zwischen Zeilennummer und dem ersten Befehl vom Computer »geschluckt« (nicht im Speicher abgelegt). Deshalb ist es ja auch möglich, statt »100 REM« zum Beispiel »100 REM« oder nur »100REM« einzugeben – nach LIST wird immer »100 REM« ausgegeben werden.

Der Grund dafür ist, daß kein einziges Leerzeichen zwischen Zeilennummer und Befehl (sonst natürlich schon) in den Speicher übernommen wird. Beim LISTen aber fügt die LIST-Routine automatisch nach der Zeilennummer genau ein Leerzeichen ein.

Jetzt wissen wir zwar, warum trotz des geLISTeten Leerzeichens auf die Zeilennummer im Speicher nicht der ASCII-Code eines Leerzeichens folgt, uns ist aber noch nicht klar, was das $8F-Byte bedeuten soll. Dieses Byte ist nämlich die Lösung unseres Ausgangsproblems (wie denn das REM gespeichert wird): $8F steht stellvertretend für REM.

Der Vorteil von $8F gegenüber $52 $45 $4D (das sind die ASCII-Codes von REM) ist, daß es sich nur um ein Byte (statt um drei) handelt und der Computer dadurch viel schneller arbeiten kann. Außerdem verringert sich durch diese Codierung – statt eines Befehlswortes wird ein Byte gespeichert – als Nebeneffekt der Speicherplatzbedarf des Basic-Programms.

Einen solchen Code für einen Basic-Befehl nennt man »Token«. Ein Token besteht in der Regel aus einem Byte, beim C 128 haben manche Befehle sogar ein »erweitertes Token«, worauf wir noch zu sprechen kommen werden.

Bei der Eingabe einer Basic-Zeile werden alle Befehlswör-

ter – sofern sie nicht nach einem REM oder in Anführungszeichen stehen – sofort codiert und in Token umgewandelt. Diese Codierung nennt man »Token-Bildung« oder »Tokenisierung«.

Ein-Byte-Token bestehen aus Werten kleiner $80. Zwei-Byte-Token (erweiterte Token) haben ein »Vor-Token«, nämlich $CE oder $FE, das das Token einer bestimmten Gruppe von Befehlswörtern zuweist. Das diesem Vor-Token folgende Byte ist eine Zahl größer als $01, die zusammen mit dem Vor-Token eine genaue Bestimmung ermöglicht.

Als Bild 25 finden Sie eine Tabelle der C128-Token.

Was wird als ASCII-Code gespeichert?

Nachdem wir nun wissen, wie Zeilennummern und Befehlswörter (wozu wir auch Funktionen und Operanden wie +, -, / und * zählen) im Speicher abgelegt werden, stellt sich die Frage, was mit den anderen Zeichen geschieht (zum Beispiel Leerzeichen, die nicht zwischen Zeilennummer und erstem Befehl stehen). Diese werden einfach im ASCII-Code abgelegt. Gleiches gilt für alle Zeichen, die entweder einem REM-Befehl folgen oder in Anführungszeichen stehen.

Am Ende einer jeden Basic-Zeile steht ein $00-Byte. Diese Endmarkierung ist unbedingt erforderlich, da die wichtigsten Unterroutinen des Basic-Interpreters von diesem Umstand ausgehen.

Nun ist uns auch klar, warum am Ende eines Basic-Programms 3 mal $00 steht: Einmal steht es als Endmarkierung der letzten Zeile und zweitens als letzter Linkpointer (also auch als Endmarkierung).

Nachdem uns nun die wichtigsten Grundlagen bekannt sind, dürften Sie in der Lage sein, Byte für Byte das Memory-Dump (Bild 25) zu erläutern. Dabei können Sie sich durchaus auch auf das Basic-Listing stützen, denn Sie müssen nicht selbst die Token umrechnen können. Sie sollten aber erkennen, ob es sich um Linkpointer, Zeilennummer, Befehlstoken oder ASCII-Code handelt.

Wenn Sie ein tiefgreifendes Verständnis gewonnen haben, wird es für Sie keine Schwierigkeit mehr sein, Manipulationen am Programm durchzuführen. Damit werden sich die nächsten Abschnitte beschäftigen.

Speichern Sie spätestens jetzt das Programm »Versuchskaninchen«, denn nun wollen wir daran Manipulationen durchführen; zwar ist die Abtipparbeit gering, aber ein Verlust des Programms aufgrund eines vielleicht kleinen Fehlers wäre ärgerlich.

Befehl	Token	Befehl	Token	Befehl	Token	Befehl	Token
END	$80	-	$AB	TRAP	$D7	TEMPO	$FE $05
FOR	$81	*	$AC	TRON	$D8	MOVSPR	$FE $06
NEXT	$82	/	$AD	TROFF	$D9	SPRITE	$FE $07
DATA	$83	↑	$AE	SOUND	$DA	SPRCOLOR	$FE $08
INPUT#	$84	AND	$AF	VOL	$DB	RREG	$FE $09
INPUT	$85	OR	$B0	AUTO	$DC	ENVELOPE	$FE $0A
DIM	$86	>	$B1	PUDEF	$DD	SLEEP	$FE $0B
READ	$87	=	$B2	GRAPHIC	$DE	CATALOG	$FE $0C
LET	$88	<	$B3	PAINT	$DF	DOPEN	$FE $0D
GOTO	$89	SGN	$B4	CHAR	$E0	APPEND	$FE $0e
RUN	$8A	INT	$B5	BOX	$E1	DCLOSE	$FE $0F
IF	$8B	ABS	$B6	CIRCLE	$E2	BSAVE	$FE $10
RESTORE	$8C	USR	$B7	GSHAPE	$E3	BLOAD	$FE $11
GOSUB	$8D	FRE	$B8	SSHAPE	$E4	RECORD	$FE $12
RETURN	$8E	POS	$B9	DRAW	$E5	CONCAT	$FE $13
REM	$8F	SQR	$BA	LOCATE	$E6	DVERIFY	$FE $14
STOP	$90	RND	$BB	COLOR	$E7	DCLEAR	$FE $15
ON	$91	LOG	$BC	SCNCLR	$E8	SPRSAV	$FE $16
WAIT	$92	EXP	$BD	SCALE	$E9	COLLISION	$FE $17
LOAD	$93	COS	$BE	HELP	$EA	BEGIN	$FE $18
SAVE	$94	SIN	$BF	DO	$EB	BEND	$FE $19
VERIFY	$95	TAN	$C0	LOOP	$EC	WINDOW	$FE $1A
DEF	$96	ATN	$C1	EXIT	$ED	BOOT	$FE $1B
POKE	$97	PEEK	$C2	DIRECTORY	$EE	WIDTH	$FE $1C
PRINT#	$98	LEN	$C3	DSAVE	$EF	SPRDEF	$FE $1D
PRINT	$99	STR$	$C4	DLOAD	$F0	QUIT	$FE $1E
CONT	$9A	VAL	$C5	HEADER	$F1	STASH	$FE $1F
LIST	$9B	ASC	$C6	SCRATCH	$F2	FETCH	$FE $21
CLR	$9C	CHR$	$C7	COLLECT	$F3	SWAP	$FE $23
CMD	$9D	LEFT$	$C8	COPY	$F4	OFF	$FE $24
SYS	$9E	RIGHT$	$C9	RENAME	$F5	FAST	$FE $25
OPEN	$9F	MID$	$CA	BACKUP	$F6	SLOW	$FE $26
CLOSE	$A0	GO	$CB	DELETE	$F7	POT	$CE $02
GET	$A1	RGR	$CC	RENUMBER	$F8	BUMP	$CE $03
NEW	$A2	RCLR	$CD	KEY	$F9	PEN	$CE $04
TAB<	$A3	JOY	$CF	MONITOR	$FA	RSPPOS	$CE $05
TO	$A4	RDOT	$D0	USING	$FB	RSPRITE	$CE $06
FN	$A5	DEC	$D1	UNTIL	$FC	RSPCOLOR	$CE $07
SPC<	$A6	HEX$	$D2	WHILE	$FD	XOR	$CE $08
THEN	$A7	ERR$	$D3	BANK	$FE $02	RWINDOW	$CE $09
NOT	$A8	INSTR	$D4	FILTER	$FE $03	POINTER	$CE $0A
STEP	$A9	ELSE	$D5	PLAY	$FE $04		
+	$AA	RESUME	$D6				

Bild 25. Die Token des C128

Da wir alles sofort ausprobieren wollen und Sie somit andauernd am Memory-Dump des »Versuchskaninchens« etwas ändern werden, vereinbaren wir vorher, wie wir Ihnen die Änderungen am Memory-Dump angeben.

Nehmen wir ein Beispiel (nur fiktiv, nicht eingeben oder testen):

>01C00 45 46 47 48 49 50 54 45 53 54...

Dies bedeutet für Sie:

- M 01C00 eingeben, sofern entsprechende Zeile noch nicht am Bildschirm ist
- die nicht-unterstrichenen Bytes unverändert lassen
- an unterstrichenen Stellen neue Werte eingeben (sechstes bis achtes und zehntes Byte)
- die Punkte »...« zeigen an, daß die dem letzten abgedruckten Wert folgenden Bytes nicht mehr geändert werden sollen

Am einfachsten ist es, wenn Sie die angezeigte Zeile direkt eingeben (natürlich ohne die Punkte am Ende). Wie gesagt: Dieses Beispiel sollen Sie noch nicht ausführen.

Ab jetzt sind die angezeigten Änderungen aber unbedingt einzugeben. Und nun viel Spaß mit den im folgenden verratenen Tips & Tricks, die Ihnen sicher zusagen werden.

Linkpointer manipulieren

Wie wir wissen, zeigt immer der Linkpointer einer Zeile auf den Linkpointer der nächsten Zeile und so weiter, bis der Linkpointer aus zwei Null-Byte, der Endmarkierung, besteht.

Die LIST-Routine benötigt diese Linkpointer, um zu wissen, ab welcher Adresse weitergeLISTet werden soll oder ob bereits das gesamte Programm geLISTet wurde.

Etwas, worauf die LIST-Routine nicht vorbereitet ist, sind Linkpointer, die aufgrund einer Manipulation mit dem Monitor nur auf sich selbst zeigen. Wenn man so etwas eingibt, wird ein und dieselbe Zeile immer wieder geLISTet (Endlos-Listing). Setzen wir unsere raffinierte Idee gleich in die Tat um:

>01C00 00 01 1C... (das dritte Byte ist sicher schon 1C)

Wie Sie sehen, steht der Linkpointer bei $1C01. Wir haben ihn mit »01 1C«, also $1C01, belegt. Um das verblüffende Ergebnis zu sehen, geben Sie doch einmal X und danach LIST ein: Die Zeile 100 (also die erste Zeile) wird immer wieder geLISTet, die Zeilen 110-140 werden »verschluckt«.

Geben Sie jetzt doch einmal »LIST 110-« ein. Der Computer – genauer gesagt: der Basic-Interpreter – verliert sich jetzt in einer Endlosschleife, denn bei der Suche nach einer bestimmten Zeile im Basic-Speicher stützt sich der Computer auf die Linkpointer. Da er die Suche mit dem ersten Linkpointer beginnt, prüft er immer nur die erste Zeile (Zeile 100) darauf, ob sie die Zeilennummer 110 hat. Diese Bedingung ist natürlich nie erfüllt, und der Computer wird »an der Nase herumgeführt«, bis man ihn mit <RUN/STOP+RESTORE> zur Vernunft bringt. Das gleiche Problem tritt auch bei GOTO, GOSUB, RUN Zeilennummer auf. Für den Fall, daß ein Programm (wie unser Beispielprogramm) überhaupt keinen solchen Befehl, der eine Zeilennummer als Argument hat, verwendet, wäre diese Linkpointer-Manipulation eine gute Listschutz-Möglichkeit, wenn nicht... Ja, wie Sie der Bemerkung »wenn nicht« entnehmen können, steht dem ein unüberwindbares Hindernis im Weg: Wir können zwar im Speicher solche Manipulationen durchführen, aber wenn der Listschutz wirksam sein soll, so muß er auch nach dem Speichern und einem Neuladen bestehen.

Wenn auch beim Speichern die manipulierten Linkpointer mitgeSAVEt werden, so wird nach dem Laden mit LOAD oder DLOAD automatisch eine Linkpointer-Neuberechnung durchgeführt, die die Linkpointer wieder in den Normalzustand versetzt – und der schöne Listschutz ist dahin.

Diese Linkpointer-Neuberechnung kann man übrigens auch über den Befehl »BANK15:SYS20303« ausführen lassen. Bei 20303 in Bank 15 steht nämlich die Routine des Basic-Interpreters zur Neuberechnung.

Unsere Manipulationen dienen also zunächst einmal nur der Demonstration; wirklich als Listschutz verwendbare Verfahren werden Ihnen selbstverständlich auch vorgestellt werden, und zwar mehr als genug!

Es geht zunächst einmal nur darum, daß Sie Praxis im Umgang mit dem Monitor bekommen und daß Sie das Prinzip verstehen.

Der genannte Befehl »BANK15:SYS20303« eignet sich aber gut, wenn man ohne Speichern und nochmaliges Laden die Versuchsmanipulationen beheben möchte.

Geben Sie deshalb jetzt diesen Befehl ein, damit das Programm wieder im Ausgangszustand ist und wir eine weitere – ähnliche – Manipulation durchführen können, die mehr ein interessanter Effekt als ein Listschutz ist.

Wir wollen dazu den letzten Linkpointer auf den Programmbeginn ($1C01) stellen, damit immer wieder das Programm geLISTet wird – nach dem LISTen der letzten Zeile soll wieder die erste ausgegeben werden und so weiter.

Verfolgen wir also die Linkpointer:

der erste (bei $1C01) zeigt auf $1C24,
$1C24 zeigt auf $1C37,
$1C37 zeigt auf $1C5A,
$1C5A zeigt auf $1C63,
$1C63 zeigt auf $1C69,
bei $1C69 steht die Linkpointer-Endmarkierung: 00 00

Der letzte wirkliche Linkpointer (also abgesehen von der Endmarkierung) ist folglich $1C63. Diesen stellen wir nun auf den Anfang des Programms:

>01C60 2C 31 00 01 1C

Das zweite zu ändernde Byte ist ohnehin schon mit $1C belegt.

Geben Sie nun LIST ein, so wird das Programm ununterbrochen gelistet (diesmal alle Zeilen), ein Abbruch ist nur mit <RUN/STOP> möglich. Wenn Sie nun eine (nicht existente) Zeilennummer 140 LISTen oder anspringen wollen, so befindet sich der Computer wieder in einer Endlosschleife, denn er ist nicht in der Lage, das Programmende zu erkennen und sucht vergeblich weiter. Aber – wie die Eingabe »BANK15:SYS20303« zeigt – ist auch dieser Spaß nicht »wetterfest«: nach dem Laden eines derart behandelten Programms geht der Effekt verloren.

Als letzte Linkpointer-Manipulation wollen wir bewirken, daß Zeilen für LIST nicht vorhanden sind, aber trotzdem abgearbeitet werden. Leider ist auch eine solche Änderung nicht gerade dauerhaft, aber zur Demonstration ist das eine nette Spielerei.

Sehen wir uns noch einmal die »Marschroute« der Linkpointer an:

1C01-1C24-1C37-1C5A-1C63-Endstation bei 1C69

Wir wollen nun die Linkpointer etwas umstellen:

1C01-1C63-Endstation bei 1C69

Dazu müssen wir einfach den Linkpointer 1C01 auf 1C63 stellen, alles weitere ergibt sich daraus von selbst:

>01C01 63 1C...

Nach diesen Manipulationen, die durch Speichern und Neuladen zerstört werden und somit kein Schutz sind, wollen wir nun auch Änderungen durchführen, die dauerhafter sind.

Zeilennummern manipulieren

Im Gegensatz zu den Linkpointern werden die Zeilennummern nicht nach jedem Ladevorgang neu berechnet, sondern bleiben unverändert. Somit sind Manipulationen an den Zeilennummern wesentlich dauerhafter als Linkpointer-Manipulationen.

Wie wir wissen, darf bei der Eingabe einer Zeilennummer die Zeilennummer nicht größer als 63999 (= $F9FF) sein. Da die Zeilennummer im Speicher als Zwei-Byte-Integerzahl im Low-High-Format abgelegt wird, wären eigentlich Zeilennummern bis (2^{16})-1 = $FFFF = 65535 möglich. Warum der Interpreter hier engere Grenzen setzt als nötig ist nicht bekannt.

Was dieses Thema für uns so interessant macht, ist die Tatsache, daß Zeilen mit Zeilennummern über 63999 nicht angesprungen und vor allem nicht über Eingabe von Zeilennummer und <RETURN> oder »DEL Zeilennummer« gelöscht werden können.

Wenn wir also Zeilen zur Verfügung haben, die nicht angesprungen werden (zum Beispiel Einzeiler-Programme), können wir mit dem Monitor auch Zeilennummern bis 65535 erzeugen. Nehmen wir wieder unser Beispielprogramm und probieren es gleich aus:

```
>01C03 E8 FD...
```

Damit haben wir die Zeilennummer der ersten Zeile (ursprünglich 100) auf $FDE8 = 65000 gesetzt. Nun setzen wir auch noch die anderen Zeilennummern neu:

```
>01C26 F2 FD...
>01C39 FC FD...
>01C5C 06 FE...
>01C65 10 FE...
```

Nach diesen Änderungen hat das Programm die Zeilennummern 65000 bis 65040. Versuchen Sie doch einmal, eine Zeile zu löschen (aber nicht mit NEW):

65000 <RETURN> ergibt einen »?SYNTAX ERROR«.

Bei den anderen Zeilennummern ist dies, weil alle größer als 63999 sind, auch nicht anders.

Da die Zeilennummern erhalten bleiben, können Sie das Programm in diesem Zustand speichern, und nach dem Laden wird es immer noch die überdimensionalen Zeilennummern haben.

Auch ein »BANK15:SYS20303« (Linkpointer-Neuberechnung) ändert nichts daran.

Der große Nachteil von derartig manipulierten Programmen ist, daß das Ändern einer Zeile mit unerlaubter Zeilennummer nicht so leicht möglich ist (mit dem Basic-Editor geht es jedenfalls nicht); dies kann aber, wenn man sein Programm gegen allzu leichte Eingriffe schützen will, ein Vorteil sein.

Besonders interessant sind überdimensionale Zeilennummern für Einzeiler, die unter Umständen nur die Basic-Start-Zeile eines Maschinenprogramms sind.

Eine andere Möglichkeit der Manipulation von Zeilennummern ergibt sich daraus, daß man eine Zeilennummer als Zeilennummer für mehrere Zeilen verwendet. Damit kann man ein Listing ziemlich chaotisch gestalten. Natürlich kann diese Mehrfach-Zeilennummer auch noch zusätzlich überdimensional sein. Damit Sie auch dies gleich in der Praxis einmal anwenden können, wollen wir in unserem Beispielprogramm jeder Zeile die Zeilennummer 65535 geben.

Dazu sind folgende Änderungen erforderlich:

```
>01C03 FF FF
>01C26 FF FF
>01C39 FF FF
>01C5C FF FF
>01C65 FF FF
```

Es ist selbstverständlich auch möglich, nur Teilen eines Programms gleiche Zeilennummern zu geben; achten sollte man nur darauf, daß an eine mehrfach verwendete Zeilennummer kein Sprung (GOTO, GOSUB, RUN) erfolgt. Wenn man sich daran hält, kann eigentlich nichts passieren.

Nun wollen wir zum Abschluß dieses Abschnitts noch Anwendungen des H-Befehls besprechen, von denen die eine bei der Manipulation eines Basic-Programms mit dem Monitor nützlich ist, die andere bei der Basic-Programmierung.

a) Suche nach einer bekannten Zeile im Speicher

Oft kommt man in die Lage, daß man eine bestimmte Zeile, deren Zeilennummer man kennt, im Speicher suchen möchte, um sie dann mit dem Monitor zu bearbeiten; insbesondere in den folgenden Abschnitten werden solche Anwendungsfälle auftreten.

Die gezielte Suche nach dieser Zeile ist für uns, da wir ja den Aufbau einer Basic-Zeile im Speicher kennen, keine Schwierigkeit mehr.

Wir müssen nur die Zeilennummer ins hexadezimale Low-High-Format umrechnen und danach gezielt suchen lassen.

Auch an dieser Stelle möchten wir Ihnen ein schrittweises Verfahren vorschlagen:

I. Umwandlung der dezimalen Zeilennummer ins hexadezimale Zahlensystem

Nehmen wir als Beispiel die Zeile 4000 (dezimal). Also müßten wir »+4000« eingeben, um das hexadezimale Ergebnis ($0FA0) zu erhalten.

II. Umwandlung der hexadezimalen Zahl ins Low-High-Format

Die beiden ersten Stellen ergeben das HB, die beiden letzten das LB. Im Beispiel: LB=$A0, HB=$0F. Das Low-High-Format ist folglich »A0 0F«.

III. Suche nach Low-High-Format der Zeilennummer

Nun ist noch der Bereich, in dem das Basic-Programm liegt, zu durchsuchen. Die Anfangs- und Endadresse kann man, wenn man diese Parameter noch nicht kennt, mit Hilfe des M-Befehls ermitteln (siehe Abschnitt über den M-Befehl).

Nehmen wir an, die Anfangsadresse ist $1C01 (Normaleinstellung) und die Endadresse $789F. Der Suchbefehl lautet dann »H 1C01 789F A0 0F«.

IV. Auswertung des Suchergebnisses

In den meisten Fällen wird man wohl nur eine Adresse als Ergebnis erhalten (nehmen wir als Beispiel $3A61). Von dieser Adresse, die die Adresse der Zeilennummer im Speicher ist, kann man dann alle zur gesuchten Zeile benötigten Adressen ausrechnen:

Die Linkpointer stehen bei Adresse-2 = $3A61-2 = $3A5F, die Tokens und ASCII-Codes der Zeile bei Adresse+2 = $3A61+2 = $3A63.

Wenn man mehrere Adressen als Ergebnis der Suche über den H-Befehl erhält, so muß man mit »M Adresse« prüfen, ob ab Adresse+2 die Codes der richtigen Basic-Zeile stehen. Dieses Prüfen von Hand ist nur selten nötig, und man erhält in den seltensten Fällen mehrere Adressen, die zu testen sind. Wenn dies dennoch der Fall ist, so kann man sich das Prüfen dadurch erleichtern, daß man zwischenzeitlich den Monitor verläßt, »LIST Zeilennummer« eingibt und dann im Monitor direkt mit der – noch am Bildschirm stehenden – geLISTeten Zeile vergleicht. So findet man die Adresse einer Programmzeile im Speicher ziemlich schnell.

b) Suche nach einer Stelle im Programm

Es gibt aber auch eine etwas andere Suche nach einer Stelle im Programm, für die es beim ansonsten überzeugenden Basic 7.0 keinen Befehl gibt: die Suche nach Programmzeilen, in denen bestimmte Stellen (zum Beispiel ein Text) vorkommen.

Dieser Anwendungsfall ist beim Basic-Programmieren sehr häufig, vor allem dann, wenn man Programme ansehen oder gar ändern will.

Auch für diesen Fall können wir auf den H-Befehl des TEDMON zurückgreifen, wobei man ungefähr folgendermaßen vorgeht:

I. Suche nach Stelle im Basic-Speicher

Hierbei ist zu beachten, daß Basic-Befehlswörter zu Token werden. Wie man Anfangs- und Endadresse des Basic-Programms im Speicher ermittelt, wissen wir bereits.

II. gefundene Stellen mit M ansehen

Der Computer gibt nach dem H-Befehl alle die Stellen aus,

an denen die Stelle steht. Da wir zur weiteren Bearbeitung über LIST und den Basic-Editor aber die Zeilennummer – diese steht im Speicher vor dem Programmtext – benötigen, lassen wir das Memory-Dump einfach bis zu 250 Byte früher als die über H gefundene Adresse beginnen. Wenn wir also $F3457 erhalten, dumpen wir »M F335A«.

III. Zeilennummer suchen

Nun suchen wir das letzte Null-Byte (Endmarkierung einer Basic-Zeile) vor der (über H) ausgegebenen Adresse. Diese Markierung ist die Endmarkierung der letzten Basic-Zeile vor der Zeile, in der der gesuchte Text steht. Auf das Null-Byte folgt dann ein für diese Suchzwecke uninteressanter Linkpointer, hinter dem schließlich die Zeilennummer im Low-High-Format steht. Mit Hilfe des $-Befehls kann man diese Zeilennummer ins dezimale Format umwandeln lassen, und einer weiteren Bearbeitung der Zeile über LIST und den Basic-Editor steht nun nichts mehr im Wege.

ASCII-Codes in REM-Zeilen unterbringen

Als Vorübung für dieses Thema, das unser letztes sein wird (es erstreckt sich über diesen und die folgenden zwei Abschnitte), sollten Sie den Abschnitt über die ASCII-Codes gelesen haben.

Wir wollen nun daran gehen, bestimmte ASCII-Codes in ein Basic-Programm einzubauen (unser Beispielprogramm Versuchskaninchen verwenden wir weiter), um Chaos oder aber Ordnung in ein Basic-Listing zu bringen. Wir werden sowohl das Listing übersichtlicher gestalten als auch Listschutzverfahren anwenden.

Wir werden mit dem Monitor (vor allem dem M-Befehl) in REM-Zeilen – damit das Programm weiterhin lauffähig bleibt – ASCII-Codes schreiben, die dann beim LISTen ausgegeben werden und somit zur Ausführung kommen.

Damit Sie – ohne große Vorkenntnisse, die wir natürlich auch erarbeiten wollen – einen Abglanz der uns erwartenden Möglichkeiten sehen können, geben Sie bitte folgenden Befehl an den Monitor ein:

>01C06 13 12

$13 (#19) ist der ASCII-Code für HOME, $12 (#18) der für RVS ON.

Wenn Sie nun LISTen (geben Sie einfach nach X den Basic-Befehl LIST ein), so erscheint der Inhalt der ersten REM-Zeile (Zeile 10) unabhängig von der Startposition des Cursors beim Listen links oben am Bildschirm (also von der HOME-Position an). Dies wird natürlich durch den über > eingebauten Code $13 ausgelöst.

Außerdem erscheint der Text »PROGRAMM VERSUCHSKANINCHEN« nun in inverser Darstellung, was durch den Code $12 verursacht wird.

Dies sollte nur ein kleiner Einstieg sein; probieren Sie doch einmal die folgenden Eingaben aus, wobei Sie vor jeder Änderung das Programm noch einmal laden sollten (damit die alten Manipulationen entfernt werden):

>1C06 1B 58
>1C06 07
>1C06 0D
>1C06 1F
>1C06 05
>1C06 1B 4A

Bei der ersten Eingabe könnte vielleicht der Eindruck entstehen, der Computer würde zum Abstürzen gebracht; dies ist jedoch nicht so: es wird durch die Steuerzeichen 1B 58 nur bewirkt, daß die Anzeige vom 40- auf den 80-Zeichen-Bildschirm wechselt oder umgekehrt.

Bei den Befehlen, bei denen auf 1C06 nur ein Byte folgt, wird im übrigen das Leerzeichen vor »PROGRAMM VERSUCHSKANINCHEN« überschrieben, weshalb es von da an nicht mehr existiert und als unmittelbare Folge nicht gelistet wird (statt dessen unser Steuerzeichen).

Auch wenn Sie nun schon ein paar weitere Beispielmanipulationen durchgeführt haben: das ist nur die Spitze des Eisbergs.

Vom Rest trennt uns aber nicht mehr viel, denn der Monitor ist mittlerweile für uns ein hilfsbereiter Freund geworden (hätten Sie das in Ihrer Anfangszeit mit dem Monitor gedacht?) und wird uns keine Schwierigkeiten machen. Wir müssen nur noch den Basic-Interpreter überlisten, und um es gleich zu sagen: das ist nicht schwer.

Einen Großteil wissen wir schon: Uns ist bekannt, wie ein Basic-Programm im Speicher abgelegt wird. Sehr interessant ist dabei die Tatsache, daß Texte nach REM-Befehlen direkt im ASCII-Code abgelegt werden.

Nun müssen wir noch wissen, wie der LIST-Befehl den Interpreter-Code (Token) des Basic-Programms in den leichtverständlichen Klartext umwandelt. Dazu geht er folgendermaßen vor:

- Zeilennummern werden ausgegeben, Linkpointer ignoriert (für den Basic-Programmierer sind diese uninteressant und müssen nicht ausgegeben werden).
- Token werden daran erkannt, daß sie größere Werte als $7F sind und nicht zwischen zwei Anführungszeichen stehen (sonst wären Werte größer als $7F als ASCII-Codes für Grafikzeichen etc. unbrauchbar); solche als Token erkannten Werte werden mit Hilfe einer Tabelle im ROM, in der die Klartexte der Befehle aufbewahrt sind, umgewandelt. Diese Tabelle haben wir uns bereits mit dem M-Befehl angesehen (M F4417).
- Nicht-Token und alle Codes, die in Anführungszeichen stehen, werden unmittelbar wie über »PRINT CHR$(Code);« ausgegeben – also ohne vorheriges Codieren oder Umwandeln in einen dazugehörigen Klartext, wie dies etwa bei Token erfolgt.
- Am Ende einer Zeile, welche anhand der Endmarkierung $00 erkannt wird, wird das Steuerzeichen RETURN (ASCII-Code $0D= #13) ausgegeben. In Basic ist dies mit »PRINT CHR$(13);« oder einfach »PRINT« (ohne Parameter) möglich.

Aufgrund der Token ergibt sich nun ein kleines Problem: Steuerzeichen mit Codes über $7F – wie zum Beispiel $93, den Code für CLR/HOME, können wir zwar einbauen – zum Beispiel über >1C06 93, aber diese werden nicht einfach an die Ausgaberoutine des Betriebssystems gesendet und dann ausgeführt, sondern als Token behandelt, weshalb anstelle des ursprünglich vorgesehenen Steuerzeichens dann ein Basic-Befehl erscheint (probieren Sie es aus: >1C06 93 schreibt zwar den ASCII-Code des Steuerzeichens für Bildschirm-Löschen unmittelbar hinter den REM-Befehl, beim Listen erscheint jedoch ein Basic-Befehl).

Der Einfachheit halber werden wir fürs erste nur Codes, die kleiner als $80 sind, einbauen; für die anderen finden wir auch noch eine Lösung, wenn wir schon mehr Übung gewonnen haben.

Das Steuerzeichen RETURN

Eine kleine Abhandlung über das ungemein wichtige Steuerzeichen RETURN wurde bis zu dieser Stelle aufgeschoben, aber jetzt wird es Zeit, daß wir uns über dieses Gedanken machen.

RETURN werden wir fortan als CR bezeichnen. CR ist die Abkürzung für »carriage return«, was man mit »Wagenrücklauf« übersetzt.

CR werden wir deshalb sagen, weil RETURN zu Verwechslungen mit dem Basic-Befehl gleichen Namens führen kann – auch wenn der Sinnzusammenhang dem Fortgeschrittenen eine klare Unterscheidung ermöglichen dürfte.

Ein CR ($0D= #13) bewirkt bei der Ausgabe – unabhängig von jedem Ausgabe-Modus – folgendes:
- Es wird auf den Normal-Modus (siehe Eingabemodi des Betriebssystems) zurückgeschaltet.
- Der Cursor springt an den Anfang der nächsten Zeile.
- Der RVS-Modus wird ausgeschaltet.

Insbesondere der erste Punkt ist sehr wichtig. Dem CR ähnlich ist der Code $8D = #141, der per Tastatur über <SHIFT+RETURN> erreicht wird. Dieser entspricht in den genannten Wirkungen exakt dem CR-Code, gilt aber nicht als Abschluß einer Eingabe. Eine vom Basic-Befehl INPUT geforderte Eingabe wird nur durch Drücken von <RETURN> (also CR) abgeschlossen, nicht über <SHIFT+RETURN>.

Für den Anfang haben wir jetzt die theoretischen Grundlagen erarbeitet; wir wollen nun in der Praxis Manipulationen durchführen, um Steuerzeichen in unser Beispielprogramm einzubauen. Hierbei verwenden wir zunächst nur die beiden REM-Zeilen, wobei in der ersten (Zeile 100) der Text erhalten bleiben soll.

Die Änderungen werden wir so angeben, daß sie nach und nach durchgeführt werden und eine Manipulation – sofern dies möglich ist – keine vorhergehende Manipulation rückgängig macht. Falls dem nicht so ist, werden wir extra darauf hinweisen.

Wir werden es Ihnen ebenfalls mitteilen, wenn unsere ersten Versuche fertig sind; das Programm befindet sich dann im manipulierten Zustand, und Sie können es durchaus einmal speichern.

Auf die Liste der Steuerzeichen des C 128 aus dem Abschnitt über die ASCII-Codes sei noch einmal hingewiesen. Nun aber zu den Manipulationen am Basic-Programm.

CR und/oder LF

CR haben wir in diesem Abschnitt schon besprochen. Neu ist jedoch die Abkürzung »LF«. LF steht für »line feed«, auf deutsch »Zeilenvorschub«. Benutzer eines Druckers dürften dieses Steuerzeichen vielleicht schon kennen. Am Bildschirm entspricht es der Taste <CRSR DOWN>.

Während CR in jedem Modus funktioniert und gewisse Einstellungen wie das Umschalten in den Normal-Modus (Anführungszeichen-Modus aus) vornimmt, tut dies der LF-Code (zu erreichen über die <LINE FEED>-Taste der C 128-Tastatur oder <CTRL+J>) nicht. Wie gesagt, er entspricht eigentlich der Bewegung CRSR DOWN.

Wir wollen nun mit Hilfe des CR-Codes erreichen, daß die Zeile
100 REM "PROGRAMM VERSUCHSKANINCHEN"
fortan so geLISTet wird:
100 REM
"PROGRAMM VERSUCHSKANINCHEN"

Dazu ändern wir das Leerzeichen zwischen REM und »PROGRAMM...« auf den CR-Code:
>01C00 00 24 1C 64 00 8F 0D 22 50 52 4F 47...

Sie können auch der Einfachheit halber >1C06 0D eingeben.

Verlassen Sie den Monitor mit X und geben Sie LIST ein, so können Sie das Ergebnis sehen.

Dabei wollen wir es jedoch nicht belassen. Ein schönerer Effekt ist es, wenn folgende Ausgabe auf LIST erscheint:
100 REM
"PROGRAMM VERSUCHSKANINCHEN"

Wie Sie sicher erkennen können, müssen wir dazu den LF-Code ($0A= #10) verwenden:
>01C00 00 24 1C 64 00 8F 0A 22 50 52 4F 47...
oder einfach >1C06 0A eingeben. Dadurch wird der CR-Code mit dem LF-Code überschrieben.

Sowohl diese Änderung als auch alle noch folgenden Manipulationen sollten Sie, um Erfahrung zu gewinnen, gleich über X und LIST ansehen, und danach wieder mit MONITOR (<SHIFT+F7>) in den Monitor zurückspringen.

Es stehen zwar 16 Farbcodes zur Verfügung, aber leider sind nur 4 davon auf ASCII-Codes gelegt, die kleiner als $80 sind:
weiß ist $05 (#05)
rot ist $1C (#28)
grün ist $1E (#30)
blau ist $1F (#31)

Wir wollen nun Farbe ins Spiel – sprich: in das Listing – bringen.

Als Beispielanwendung soll nach dem LISTen von »100 REM« die aktuelle Zeichenfarbe auf blau (ASCII-Code: $1F= #31) geschaltet werden. Dazu muß in Basic CHR$(31) ausgegeben werden, in unserem Fall – dem Basic-Programm – muß der ASCII-Code des Farbsteuerzeichens in der REM-Anweisung stehen, damit er dann ausgegeben wird und sich die Zeichenfarbe ändert:
>01C00 00 24 1C 64 00 8F 0A 1F 50 52 4F 47...
oder einfach >1C07 1F eingeben.

Wir opfern also dem Farbsteuerzeichen das nicht so elementar wichtige, erste Anführungszeichen.

Nachdem wir das erste Anführungszeichen geopfert haben, ist das zweite eigentlich auch überflüssig. Denkbar wäre unter anderem, dies mit einem CR oder LF zu überschreiben. Wir wollen jedoch, daß nach der Kommentarzeile der Text weiß ausgegeben wird.

Sicher haben Sie schon erraten, was dazu getan werden muß: der Code des zweiten Anführungszeichens, der im Speicher bei $1C22 steht, muß mit dem Steuerzeichen überschrieben werden, welches die Zeichenfarbe auf weiß stellt:
>01C20 45 4E 05 00 37 1C 6E 00...
oder einfach >1C22 05 eingeben.

Sie können auch anstelle unserer Beispielfarben weiß (das war der $05-Code) und blau ($1F) andere Farbsteuerzeichen – die ASCII-Codes zur Einstellung der zunächst möglichen Farben wurden Ihnen bereits angegeben – ausprobieren.

Cursor in Home-Position bringen

Während die Farben – sofern nicht gerade die Zeichenfarbe den gleichen Wert wie die aktuelle Hintergrundfarbe erhält – eher zur Übersichtlichkeit eines Listings beitragen als zu chaotischen Ausgaben, die Unbefugten den Zutritt zum Programm verwehren oder zumindest erschweren sollen, kann man manche Codes ganz gut einsetzen, um Verwirrung im Listing zu stiften.

Wie Sie wissen, bringt die HOME-Taste den Cursor in die Home-Position, das heißt die linke obere Ecke des Bildschirms beziehungsweise Textfensters (falls man nur einmal auf <HOME> drückt und man überhaupt auf die Window-Technik zurückgreift).

Der Cursor bestimmt immer die Stelle, an die die nächste Bildschirmausgabe erfolgt. Wenn wir ihn also wie über die <HOME>-Taste in die linke obere Ecke wandern lassen, so wird das nächste Zeichen nach HOME, das geLISTet wird, links oben ausgegeben. Das Interessante daran ist, daß alle Texte, die an den betreffenden Stellen stehen, überschrieben werden, die Stellen in einer Zeile, die nicht überschrieben werden, aber nicht gelöscht werden, sondern ungestört erhalten bleiben. Wenn man nun in einem längeren Programm an mehreren Stellen (zum Beispiel alle zehn Zeilen) einen solchen »Stör-Code« anbringt, würde regelmäßig die Ausgabe wieder links oben beginnen, und das Listing der vorher geLISTeten Zeilen würde – zumindest teilweise – wieder überschrieben werden. Da jedoch Basic-Zeilen unterschiedlich lang sind und somit in geLISTetem Zustand unterschiedlich viel Platz am Bildschirm beanspruchen, bliebe ab und zu den-

noch ein Teil des Bildschirms, nämlich der, der nicht durch das Listing der neuen Zeilen überschrieben wird, weiter erhalten, und es ließe sich nicht eindeutig bestimmen, wo die gerade angezeigte Basic-Zeile endet und wo nicht.

Dies mag vielleicht nicht so einfach zu verstehen sein, wir wollen es darum einmal testen.

Dazu opfern wir in der REM-Zeile 120 das Leerzeichen zwischen dem REM-Befehl und den Sternchen »***« für den ASCII-Code von HOME ($13=#19):
>01C38 1C 78 00 8F 13 2A 2A...
oder >1C3C 13 eingeben.

Jetzt können Sie mit X und LIST den Effekt sehen: Die Sternchen aus Zeile 120 und die Zeilen 130/140 werden wieder von links oben geLISTet. Wenn Sie das Überschreiben sehen wollen, geben Sie folgenden Basic-Befehl ein:
SCNCLR:LIST

Der Befehl SCNCLR bewirkt, daß der Bildschirm gelöscht und der Cursor in die HOME-Position gebracht wird. Bei einem längeren mit HOME-Codes manipulierten Programm ist dieser SCNCLR-Befehl nicht nötig, da aufgrund des Scrollings (Abrollen des Bildschirms) früher oder später die Ausgabe oben ankommt (weil weiter unten ausgegebene Zeilen hoch-gescrollt werden).

Das akustische Klingelzeichen

Aufgrund des Steuerzeichens für das Klingelzeichen (ASCII-Code: $07=#7, drücken Sie einmal auf <CONTROL+G>, damit Sie es hören) können wir neben Farbe, Ordnung und Chaos sogar Soundeffekte ins Listing integrieren. Zwar sind diese recht bescheiden – ein einziges Tonsignal steht uns nur zur Verfügung -, aber ganz gut geeignet, um zum Beispiel sich selbst auf eine bestimmte Stelle im Listing aufmerksam zu machen.

Man muß im Grunde nur den Code $07 (#7) in einer REM-Zeile einbauen. Hierbei sind drei Dinge zu beachten:

Wenn der Code in Anführungszeichen steht, wird er – wie alle Steuerzeichen außer CR – unterdrückt.

Die Klingelzeichen-Codes sollten nicht zu kurz aufeinanderfolgen, weil es sonst sein kann, daß das eine Klingelzeichen noch verklingt, während das andere beginnen soll. Dies hätte zur Folge, daß das Klingelzeichen nicht unbedingt erfolgt.

Das akustische Klingelzeichen darf nicht vorher durch <ESC+H> verboten worden sein. Dies kann man dadurch umgehen, daß im Listing die Sequenz <ESC+G>, die das Klingelzeichen wieder zuläßt, ausgegeben wird. Damit werden wir uns noch in diesem Abschnitt – allerdings an späterer Stelle – befassen.

Da der erste Punkt ohnehin für fast alle Steuerzeichen gilt, sind also nur zwei Punkte beim Klingelzeichen zusätzlich zu beachten. Den dritten Punkt werden wir auch noch überflüssig werden lassen.

Nun aber soll das Klingelzeichen im Programm Versuchskaninchen auf Kosten eines Sternchens »*« integriert werden:
01C38 1C 78 00 8F 13 07 2A...
oder einfach >1C3D 07 eingeben.

Der Tabulator

Um komfortable Einrückungen vorzunehmen, ist die <TAB>-Taste vorgesehen. Diese kann mit Hilfe von <CONTROL+I> leicht ersetzt werden.

Der ASCII-Code ist $09 (#9), und anstelle des nächsten Sternchens soll dieser nun seinen Platz im Listing finden:
>01C38 1C 78 00 8F 13 07 09 2A...

Wenn Sie jetzt das Programm LISTen lassen, können Sie sehen, daß die Sternchen eingerückt werden.

Mit Hilfe der Tasten <SHIFT+CBM> (Commodore-Taste) kann man zwischen dem Text- und dem Grafik-Zeichensatz umschalten. Es gibt allerdings Situationen, in denen eine Umschaltung wüstes Chaos hervorrufen würde (zum Beispiel weil anstelle von Grafikzeichen nur Großbuchstaben erscheinen oder umgekehrt), besteht auch eine Möglichkeit, die Umschaltung zu unterbinden. Auch ein Entriegeln der Tastenkombination <SHIFT+CBM> ist möglich. Die ASCII-Codes dazu sind $0B (#11) zum Ver-, $0C (#12) zum Entriegeln.

Probieren wir zunächst das Steuerzeichen $0C (zum Entriegeln) aus:
>01C38 1C 78 00 8F 13 07 09 0C...
oder nur >1C3F 0C eingeben.

Verriegeln Sie nun die Umschaltung über <CONTROL+K>. LISTen Sie dann (einfach <F7> drücken) und schalten Sie dann zwischen den Zeichensätzen um. Aufgrund des Steuerzeichens in der REM-Zeile 120 wird es nun gehen – quod erat demonstrandum.

Sie stimmen mir sicher zu, daß ein Entriegeln von <SHIFT+CBM> in den meisten Fällen sinnlos ist. Viel wichtiger ist die Funktion des Verriegelns, die wir folgendermaßen – anstelle des $0C-Codes – einbauen wollen:
>01C38 1C 78 00 8F 13 07 09 0B...
oder nur >1C3F 0B eingeben.

Dieses Verbot des Zeichensatz-Umschaltens ist erst in Verbindung mit der programmierten Einstellung eines bestimmten Zeichensatzes möglich. Da der ASCII-Code zum Umschalten auf Text-Zeichensatz größer als $80 ist und das Umschalten des Zeichensatzes mittels Steuerzeichen nicht zu verhindern ist, werden wir auch einen solchen Code einbauen.

Umschalten auf Text-Zeichensatz

Der Text-Zeichensatz wird über PRINT CHR$(14) eingestellt. Das Steuerzeichen hat den ASCII-Code $0E (#14), und muß nur in die REM-Zeile eingebaut werden:
>01C40 0E 2A...
oder >1C40 0E eingeben.

Wenn Sie nun LIST eingeben, wird spätestens ab der Stelle, an der das $0E-Steuerzeichen steht, der Text in Klein-/Groß-Schrift ausgegeben.

Blinkender Text

Mit Hilfe von Steuerzeichen können wir auch bewirken, daß der Text einer bestimmten Zeile – bis eben das nächste CR ausgegeben wird – in blinkender Form erscheint. Das entsprechende Steuerzeichen zum Einschalten des Blink-Modus für bestimmte Zeichen hat den ASCII-Code $0F (#15). Diesen bringen wir auch in der REM-Zeile 120 unter:
>01C40 0E 0F 2A...
oder >1C41 0F eingeben.

Nun blinken die dem $0F-Code folgenden Sternchen beim LISTen.

Die CRSR-DOWN- und die CRSR-RIGHT-Bewegung

Auch die ASCII-Codes der CRSR-DOWN-Bewegung (Cursor runter) und der CRSR-RIGHT-Bewegung (Cursor nach rechts) liegen unter $80. Die Codes sind:
CRSR DOWN = Cursor runter $11 = #17
CRSR RIGHT = Cursor rechts $1D = #29

Testen wir zunächst den CRSR-RIGHT-Code:
>01C40 0E 0F 1D 2A...
oder >1C42 1D eingeben.

Jetzt wird vor den Sternchen einmal der CRSR-RIGHT-Code ausgeführt.

Viel häufiger wird jedoch die Anwendung des CRSR-DOWN-Codes sein, weshalb wir unsere erste Manipulation – den CRSR-RIGHT-Code – überschreiben:
>01C40 0E 0F 11 2A...
oder >1C42 11 eingeben.

Nun werden die Sternchen eine Zeile weiter unten ausgegeben als vorher.

Da durch eine Cursor-Bewegung nicht auf den Normal-Modus zurückgeschaltet wird, bleibt der Blink-Modus weiterhin erhalten.

Inverse Kommentare

Um bestimmte Texte – vor allem Kommentare in REM-Zeilen – gut hervorzuheben, ist die Invers-Darstellung sehr gut geeignet. Wir wollen nun – auch wenn ein weiterer Stern dran glauben muß – den RVS-ON-Code (ASCII-Code: $12=#18) einbauen, damit der blinkende Text zudem invers abgebildet wird:
>01C40 0E 0F 11 12 2A...
oder >1C43 12 eingeben.

Wenn man bestimmte Kommentare zwecks besserer Lesbarkeit hervorheben möchte, empfiehlt es sich wirklich, diese invers erscheinen zu lassen. Am besten setzt man vor den Kommentar, der invers werden soll, bei der Eingabe über den Basic-Editor ein Zeichen, das dann mit dem Code $12 überschrieben werden kann.

Besonders interessant ist hierbei, daß auch beim Listen auf den Drucker die Invers-Darstellung berücksichtigt wird.

In unserem Beispielprogramm wollen wir nun, daß anstelle der Sternchen ein mehr oder weniger sinnvoller Text erscheint. Wir entscheiden uns für »REVERSER KOMMENTAR !!« und wandeln diesen mit einem bereits vorgestellten Trick in den entsprechenden ASCII-Code um.

Wir geben dazu »H 0 0 'REVERSER KOMMENTAR !!« ein und finden dann die Lösung bei $0A80 mittels »M A80«.

Nun setzen wir die Lösung an der richtigen Stelle – bei $1C44 – ein:
>1C44 52 45 56 45 52 53 45 52 20 4B 4F 4D 4D 45 4E 54 41 52 20 21 21
Geben Sie diesen Befehl unverändert ein.

Nun haben wir einen ersten Satz von Änderungen durchgeführt. Am besten speichern Sie das Programm in diesem Zustand; löschen Sie aber die alte Version nicht.

Von den Steuerzeichen unter $80 haben wir bis auf zwei – DEL und ESC – schon alle behandelt und die meisten ins Programm Versuchskaninchen eingebaut. Nun befassen wir uns noch mit den fehlenden Steuerzeichen DEL und ESC, die wirklich interessante Manipulationen erlauben. Zunächst soll uns DEL interessieren.

DELETE–Zeichen löschen

Mit Hilfe der Taste <DEL/INS> kann man das letzte ausgegebene Zeichen löschen. Das Steuerzeichen DEL (DEL als Abkürzung für DELETE, englisch »löschen«) arbeitet außer im »insert mode« in jedem Modus – also auch im Anführungszeichen-Modus.

Eine häufige Anwendung von DEL (ASCII-Code: $14=#20) ist es, daß man Teile des Listings durch eine REM-Zeile voller DEL-Codes unmittelbar nach dem Ausgeben wieder löscht, so daß der Eindruck entsteht, es wäre kein Text ausgegeben worden.

Wir wollen nun testhalber bewirken, daß beim LISTen von Zeile 120 der Anfang »120 REM« unmittelbar nach dem LISTen gelöscht wird. Wir benötigen sieben DEL-Codes, da beim LISTen sieben Zeichen (120 REM) ausgegeben werden. Daß REM als Ein-Byte-Wert (Token) abgelegt wird, ist in diesem Fall uninteressant, da wir drei DEL-Codes – REM – benötigen, denn die Tokens werden ja vor dem LISTen in den Klartext umgewandelt.

Für die sieben DEL-Codes müssen wir auf ein paar Steuerzeichen verzichten:
>01C3C 14 14 14 14 14 14 14 12 52...
oder >1C3C 14 14 14 14 14 14 14 eingeben. Noch einfacher einzugeben ist der Fill-Befehl »F 1C3C 1C42 14«.

Wenn Sie jetzt »LIST 120« eingeben, sehen Sie folgendes in Inversdarstellung:
REVERSER KOMMENTAR !!

Der Text 120 REM wird nämlich durch die sieben DEL-Codes sofort wieder gelöscht. Wenn man genau hinsieht, kann man dies sogar erkennen (wenn auch nur schlecht, da das Löschen doch ziemlich schnell geht). Besonders langsam geht es, wenn man im Slow-Modus auf 80-Zeichen-Darstellung schaltet und dann LISTen läßt. In diesem Fall kann man fast jedes Zeichen erkennen.

Um den Übergang zum Rest dieses Abschnitts zu finden, laden Sie bitte noch einmal die vor dem Einfügen der DEL-Codes gespeicherte Version – oder die Anfangsversion, wenn Sie vor dem Einsetzen der DELs das Speichern vergessen haben. Geben Sie dann >1C3C 1B 44 ein und LISTen Sie. Jetzt geht das Löschen sehr schnell und wir müssen nur zwei – statt sieben – Codes dafür verwenden.

Wir haben nämlich auf eine ESC-Anweisung zurückgegriffen, die eine ganze Bildschirmzeile löscht: <ESC+D>.

Das mächtige Werkzeug ESC

ESC ist die Abkürzung für ESCAPE (englisch »fliehen«). Mit Hilfe der <ESC>-Taste (links oben an der C128-Tastatur) kann man einige Funktionen anfordern (sogenannte Flucht- oder ESCAPE-Sequenzen), die im C128-Handbuch aufgeführt sind. Diese erlauben unter anderem
- die Definition von Bildschirmfenstern
- das Erlauben/Verbieten des Klingelzeichens
- Einfügen und Löschen von Bildschirmzeilen
- Scrolling nach oben und unten
- Cursor-Bewegung an Zeilenanfang oder Zeilenende
- Löschen eines Teils einer Zeile
- Löschen des Bildschirms ab Cursor-Position

und vieles andere.

Der ESC-Code, der vom aktuellen Eingabemodus völlig unabhängig ist, arbeitet folgendermaßen:

Sobald ein ESC (ASCII-Code: $1B=#27) ausgelöst wird – zum Beispiel durch Drücken der ESC-Taste –, merkt sich dies der Computer. Das nächste ausgegebene Zeichen wird dann nicht ausgegeben, sondern als ESC-Anweisung behandelt. Diese Anweisungen werden durch einige Buchstaben und den Klammeraffen (@) bezeichnet:

A = AUTO INSERT MODE ON; automatisches Einfügen einschalten
B = BOTTOM; rechte untere Ecke des Windows setzen
C = CLEAR AUTO INSERT MODE; AUTO INSERT MODE ausschalten
D = DELETE; aktuelle Bildschirmzeile löschen
E = END CURSOR FLASH; Cursor-Blinken beenden
F = FLASH ON; Cursor-Blinken einschalten
G = <CONTROL+G> erlauben
H = <CONTROL+G> verbieten

I = INSERT; Bildschirmzeile einfügen
J = JUMP TO BEGINNING; Cursor an Anfang der Zeile setzen
K = Cursor hinter letztes Zeichen der Zeile setzen
L = LET SCROLL; Scrolling zulassen
M = Scrolling verhindern
N = NORMAL; normale Bildschirmdarstellung (nicht invers)
O = OFF; alle Modi abschalten (Blinken, Unterstreichen, Invers)
P = clear until POSITION; bis Cursor-Position löschen
Q = von Cursor-Position an aktuelle Zeile löschen
R = REVERS; Revers-Darstellung für gesamten Bildschirm
S = SWITCH ON; Blockdarstellung für den Cursor einschalten
T = TOP; linke obere Ecke des Windows setzen
U = Strichdarstellung für Cursor
V = Scrolling nach oben
W = Scrolling nach unten
X = zwischen 40- und 80-Zeichen-Darstellung umschalten
Y = voreingestellte Tabulatorstops nehmen
Z = voreingestellte Tabulatorstops löschen
@ = Bildschirm ab Cursor-Position löschen

Die Befehle N,R und U funktionieren nur im 80-Zeichen-Modus.

Wie Sie sofort sehen können, kommt eine ungeheure Vielfalt an Manipulationsmöglichkeiten auf uns zu.

Der große Vorteil des ESC-Befehls ist, daß er mit $1B (#27) einen ASCII-Code hat, der kleiner als $80 ist. Wenn wir nun einen ESC-Befehl im Speicher ablegen wollen, benötigen wir immer nur zwei Byte:

1B und ASCII-Code des Befehls (zum Beispiel $41 für A)
^
ESC-Code

Die ESC-Befehle

Wir wollen nun das Beispielprogramm Versuchskaninchen (bitte laden Sie jetzt die ursprüngliche Version ohne Manipulationen) mit ESC-Befehlen (auch »ESC-Sequenzen« genannt) bearbeiten.

Als erstes sorgen wir dafür, daß der Kommentar in Zeile 100 ohne das davorstehende »100 REM« erscheint. Wir überschreiben zu diesem Zweck das Leerzeichen und das Anführungszeichen nach dem REM mit der Sequenz ESC-D (DELETE aktuelle Bildschirmzeile löschen):

>01C00 00 24 1C 64 00 8F 1B 44 50 52 4F...
oder einfach >1C06 1B 44 eingeben.

$1B ist der ASCII-Code für ESC, $44 derjenige für die ESC-Anweisung D (DELETE).

Nun führen wir noch einige Manipulationen durch, zu denen nicht viel zu sagen ist.

Als erstes soll der Bildschirm ab Cursor-Position beim Listen von Zeile 120 gelöscht werden:

>01C38 1C 78 00 8F 1B 40 2A 2A...
oder >1C3C 1B 40 eingeben.
1B 40 entspricht ESC-@.

Als zweites soll die entsprechende Bildschirmzeile bis zur Cursor-Position gelöscht werden (damit das 120 REM wegfällt):

>01C38 1C 78 00 8F 1B 40 1B 50 2A...
oder >1C3E 1B 50 eingeben.
1B 50 entspricht ESC-P.

Ferner soll während des LISTens von Zeile 120 eine Umschaltung der Zeichendarstellung (40- oder 80 Zeichen pro Zeile) erfolgen. Dadurch wird dann der Rest des Listings auf einen anderen Monitor ausgegeben als der Beginn, was nicht zur Übersichtlichkeit des Listings beiträgt. Wenn man in längeren Programmen alle paar Zeilen umschalten läßt, wird das Listing wirklich unübersichtlich – allerdings existiert noch der Ausweg des LISTens auf den Drucker. Nun die entsprechende Änderung am Programm Versuchskaninchen:

>01C40 1B 58 2A 2A...
oder >1C40 1B 58 eingeben.
1B 58 entspricht ESC-X.

Nur als Spielerei – damit es auch einmal besprochen wird – soll noch der Bildschirm nach oben gescrollt werden:

>01C40 1B 58 1B 56 2A 2A...
oder >1C42 1B 56 eingeben.
1B 56 entspricht ESC-V.

Wenn Sie statt des Scrollings nach oben lieber nach unten scrollen wollen, ist folgende Änderung nötig (anstelle von ESC-V):

>01C40 1B 58 1B 57 2A 2A...
oder >1C42 1B 57 eingeben.
1B 57 entspricht ESC-W.

Außerdem sollen oberhalb der aktuellen Bildschirmzeile zwei Zeilen eingefügt werden:

>01C40 1B 58 1B 57 1B 49 1B 49 2A 2A...
oder einfach >1C44 1B 49 1B 49 eingeben.
1B 49 entspricht ESC-I.

Ganz nebenbei wollen wir auch noch alle Zusatz-Modi (Blinken, Invers, Unterstreichen) abstellen:

>01C48 1B 4F 2A 2A...
oder >1C48 1B 4F eingeben.
1B 4F entspricht ESC-O.

Dafür soll noch der ganze 80-Zeichen-Bildschirm, auf dem aufgrund der Sequenz ESC-X am Anfang der Zeile 120 sicher ein Teil des Listings steht, invertiert werden, wobei zum Funktionieren der entsprechenden ESC-Sequenz nicht einmal der 80-Zeichen-Modus angeschaltet sein muß:

>01C48 1B 4F 1B 52 2A 2A...
oder >1C4A 1B 52 eingeben.
1B 52 entspricht ESC-R.

Zwischendrin lassen wir das nächste Sternchen weiterexistieren (damit auch ein Sternchen »*« ausgegeben wird). Dann aber stellen wir den Cursor auf Festanzeige (das heißt, wir stellen das Cursor-Blinken ab):

>01C48 1B 4F 1B 52 2A 1B 45 2A 2A 2A...
oder >1C4D 1B 45 eingeben.

1B 45 entspricht ESC-E. Das Gegenstück wäre 1B 46 (ESC-F).

Vorhin haben wir das akustische Klingelzeichen (ASCII-Code: $07=#7) verwendet. Dabei wurde der Hinweis gegeben, daß durch eine ESC-Sequenz, nämlich ESC-H, dieses Klingelzeichen unterbunden werden kann. Unter Zuhilfenahme einer anderen ESC-Sequenz läßt sich ein solches eventuell erfolgtes Unterbinden aber leicht wieder rückgängig machen:

>01C4D 1B 45 1B 47 2A 2A...
oder >1C4F 1B 47 eingeben.

1B 47 entspricht ESC-G. Um die Wirkung davon zu testen, bietet es sich an, direkt hinter der ESC-Sequenz ein Klingelzeichen einzubauen:

>01C4D 1B 45 1B 47 07 2A 2A...
oder >1C51 07 eingeben.

$07 ist der ASCII-Code des akustischen Klingelzeichens.

Zu guter Letzt definieren wir über die Sequenz ESC-T-TAB-ESC-B ein Window von wenigen Spalten Breite und nur einer Zeile Länge:

>01C50 47 07 1B 54 09 1B 42 2A 2A...
oder >1C52 1B 54 09 1B 42 eingeben.

1B 54 entspricht ESC-T, 09 entspricht TAB und 1B 42 entspricht B.

Nun haben wir einige ESC-Sequenzen, mit denen sich viel anstellen läßt, ausprobiert.

Ergänzen Sie jetzt Ihre 64'er-Sammlung

Schaffen Sie sich ein interessantes Nachschlagewerk und gleichzeitig ein wertvolles Archiv!

Kennen Sie alle Ausgaben von 64'er? Suchen Sie einen ganz bestimmten Testbericht? Oder haben Sie einen Teil eines interessanten Kurses versäumt? Suchen Sie nach einer speziellen Anwendung?

Damit Sie jetzt fehlende Hefte mit »Ihrem« Artikel nachbestellen können, finden Sie auf diesen Seiten eine Zusammenstellung aller wesentlichen Artikel der Ausgaben 01 bis 12/85.

Und so kommen Sie schnell an die noch lieferbaren Ausgaben: Prüfen Sie, welche Ausgabe in Ihrer Sammlung noch fehlt, oder welches Thema Sie interessiert. Tragen Sie die Nummer dieser Ausgabe und das Erscheinungsjahr (z.B. 2/85) auf dem Bestellabschnitt der hier eingehefteten Bestell-Zahlkarte ein. Die ausgefüllte Zahlkarte einfach heraustrennen und Rechnungsbetrag beim nächsten Postamt einzahlen. Ihre Bestellung wird nach Zahlungseingang umgehend zur Auslieferung gebracht.

Aktuell

Stichwort	Titel	Seite	Ausgabe
Allgemeines	Commodore Gestern Heute Morgen	10	01/85
Computer	Amiga — Der neue Supercomputer	8	09/85
Interview	Interview mit David Crane (Game Designer)	146	06/85
Lernen	Schule braucht Computer (VAM-Computer)	9	06/85
Messen	International Chaos Communication Congress	15	03/85
	Heiße Messe in der Wüste: CES	8	03/85
	Hannover-Messe '85	8	06/85
	Hannover-Messe '85	8	07/85
	Chicago im Zeichen der CES	8	08/85
	Aktuelles von der C'85 in Köln	15	08/85
	Btx Total (Internationale Funkausstellung)	8	10/85
	PCW-Computermesse in London	8	11/85
	Neues von der Commodore-Fachausstellung 1985	8	12/85
Recht	Die neue Abmahnmasche — Vorsicht bei Programmangeboten	8	05/85
	Die Ex-Knacker — wo sind sie geblieben?	27	08/85
	Interview mit Raubkopierern (Section 8)	28	08/85
	Schützer kontra Knacki's	23	08/85
	Raub-Talkshow	12	08/85
	Das Urheberrechtsgesetz und Gedanken zu seiner Anwendung	21	08/85
	Änderung des Urheberrechtsgesetzes	162	09/85

Buchbesprechungen

Stichwort	Titel	Seite	Ausgabe
Anfänger	Goldmann Computer Compact	87	03/85
	Basic-Wegweiser für den C 64	86	05/85
	Alles über den C 64, Sachbuchreihe, Band 1	115	06/85
	Lehrspielzeug Computer: C 64/VC 20	112	11/85
	C 64 Computerhandbuch	171	11/85
	Einführungskurs: Commodore 64	144	12/85
Anwendung	Dienstprogramme VC 20, C 64 und SX	86	05/85
	Spaß an Mathe mit dem Commodore 64	88	07/85
	Mathe für die Oberstufe mit dem C 64	88	07/85
	Mathematische Routinen VC 20, Elektrotechnik/Elektronik	112	11/85
	Commodore 64-Listings, Band 2: Dateiverwaltung, Schule, Hobby	112	11/85
	Das Trainingsbuch zum Datamat	144	12/85
C 128	Bücher zum C 128	22	10/85
DFÜ	Das Mailbox-Jahrbuch: Nutz die Netze	112	11/85
Grafik	Grafik auf dem Commodore 64 (+Fehlert. 9/85)	86	05/85
	Einführung in CAD mit dem Commodore 64	128	06/85
	Grafik & Musik auf dem Commodore 64	88	07/85
	Verschiedene Grafikbücher zum C 64	115	08/85
Programmieren	Von Basic zu Assembler: Das Commodore-Buch, Band 4	115	06/85
	64 Intern	115	06/85
	Das Interface Age System-Handbuch zum C 64	115	06/85
	Das C 64 Buch, Band 5: Simons Basic Leitfaden	144	12/85
	Basicode	144	12/85
	Noch mehr Tips und Tricks zum 64er	144	12/85
Speichern	Das Kassettenbuch zum C 64 und VC 20	87	03/85
	Die Floppy 1541 (M&T)	88	07/85
Spiele	Rombachs C 64 Spielführer	87	03/85
	Commodore 64-Listings, Band 1, Spiele	112	11/85
	35 ausgesuchte Spiele für Ihren Commodore 64	171	1/85

64'er Extra

Stichwort	Titel	Seite	Ausgabe
Prozessor	Befehlssatz des 6502/6510-Prozessors	84	09/85
Grafik	Die Videochip-Register des C 64	92	10/85
Sound	Der SID-Chip, seine Register und Programmierung	92	11/85
Speicher	Die Speicherbelegung des C 64	96	12/85

Abenteuerlösungen

Stichwort	Titel	Seite	Ausgabe
Lösungen	Dallas-Quest Lösung	90	01/85
	Guncho Krill-Enchanter ist gelöst	44	03/85
	Infocom-Geheimnisse gelüftet?	49	05/85
	Des Rätsels Lösung: Amazon	145	06/85
	Activision-Adventures entschleiert (Mindshadow, Tracer Sanction)	36	12/85
	Eureka! — ich hab's!	37	12/85
	Lösungen zu Hitchhiker's Guide und Sorcerer	39	12/85

Spiele-Tests

Stichwort	Titel	Seite	Ausgabe
007	James Bond — A View to a Kill	156	09/85
Abenteuer	Abenteuerpaket 1	48	08/85
	Shadowfire	146	09/85
	The Quest — mit C 64 auf Suche nach Drachen	47	01/85
Action	Hexenküche	50	07/85
	Master of the Lamps	48	07/85
	Rescue on Fractalus	158	10/85
	Stellar 7	49	08/85
Construction Set	Mail Order Monsters	49	08/85
	Racing Destruction Set	50	08/85
Geschicklichkeit	Australopedicus Robustus	50	08/85
	Boulder Dash II	159	10/85
	Crystal Castles	50	07/85
	Gribbly's Day out	148	09/85
	Rock'n Bolt	48	08/85
	Thing on a Spring	159	10/85
	Tom + Zaga	48	01/85
Pseudo-Adventures	Roland's Rat Race	49	08/85
	Fourth Protocol und Frankie g.t.H.	162	11/85
Renner	Die Renner 1985: Meistverkaufte Spiele	34	12/85
Schach	Viermal Schachmatt: Verschiedene Schachprogramme	32	12/85
Simulation	Elite	148	09/85
	Jump Jet	148	09/85
	Super Huey Hubschraubersimulator	49	07/85
Sport	Boxspiele: Frank Bruno's B. + Barry McGuigan Champions. B.	49	12/85
	Handkantenschlag per Joystick: Karateka + Exploding Fist	165	11/85
	Nick Faldo Plays the Open (Golf)	159	10/85
	Rallye Speedway	49	07/85
	Slapshot (Eishockey)	50	07/85
	Summer Games II	146	09/85
	World Series Baseball	49	07/85
Diverses	New York City und Air Support	145	06/85

Hardware-Tips und Bauanleitungen

Stichwort	Titel	Seite	Ausgabe
Audio/Video	Mit 5 Mark zu neuen Dimensionen (Stereoanlage am C 64)	34	05/85
	Ein Monitor ist genug (RGB+Composite an C 128)	16	10/85
C 16	Alte Datasette am C 16	31	04/85
	Alter Joystick am C 16	35	05/85
Eingabegeräte	Der Hexer — Zusatztastatur für den MSE	48	10/85
EPROM	EPROMs im Expansion-Port	46	10/85
	EPROM-Trans — Die Super-Erweiter...	42	10/85
	Das 64'er EPROM-Programmiergerät	44	12/85
Floppy/Datasette	Diskettenlaufwerk 1541 selbst justiert	32	10/85
	Die Datasette streikt nie wieder (Anpassung des Tonkopfs)	34	10/85
IEC-Bus	Auf zu neuen Welten: IEC-Bus im Selbstbau (+Fehlerteufel 10/85)	44	07/85
Joystick	Joystick im Selbstbau	33	03/85
	Dauerfeuer-Adapter	46	08/85
RS232/V.24	Das 30-Mark-Interface (Selbstbau RS232)	29	03/85
	Genau betrachtet: Die RS232/V.24-Schnittstelle	80	05/85
Diverses	Userport-Display	36	05/85
	Reset-Taster für alle Fälle (+Fehlert. 9/85)	130	06/85
	Aus eins mach vier (absturzfreie Betriebssystemumschaltung)	41	07/85

Hardware-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Computer	Was bringt der C 128?	28	11/85
Drucker	Welcher Drucker ist der Richtige? (Grundlagen)	15	05/85
	Hammerwerke — wie funktionieren Typenraddrucker	32	06/85
	Die Alternativen: Thermo-, Tintenstrahldrucker + Plotter	24	07/85
Eingabegeräte	Versteht Sie Ihr Computer? (Wie funktionieren Eingabegeräte)	44	09/85
Floppy	Floppy oder Datasette?	129	06/85
Monitore	Wie funktionieren sie, was ist beim Kauf zu beachten?	16	12/85
	Das Kabel zum Monitor: Welche Normen gibt es?	28	12/85
Peripherie	Grafikeingabegerät: Wie funktionieren sie?	30	08/85

Hardware-Tests

Stichwort	Titel	Seite	Ausgabe
Computer	Generationswechsel: Test C 16	16	01/85
	Erster ausführlicher Test C 128 PC (Teil 1)	16	06/85
	Erster ausführlicher Test C 128, PC (Teil 2)	17	07/85
DFÜ	Marktübersicht Modems & Akustikkoppler	32	07/85
Drucker	Vergleich: Drucker unter 700 Mark (Tests und Marktübersicht)	18	05/85
	Tests und Marktübersicht Typenraddrucker	35	06/85
	Test: Brother EP 44	27	07/85
	Brother TC-600	118	08/85
	Riteman C+	133	09/85
	Panasonic KX-P1091	134	09/85
	Star SG 10C	132	09/85
	Melchers CP-80X — wie hätten Sie's denn gern?	25	10/85
	Geheimtip: Der RFI DP 165	24	10/85
	Epson GX 80 — einer für alle	26	10/85
	MPS 803 — ein Drucker für alle Gelegenheiten?	40	1/85
	Epson JX-80 das vielfarbige Druck-Genie	38	11/85
	Epson FX-85 neue Referenz	42	11/85
	SP 1000 VC — Superstar mit Haken	41	11/85
	Der NEC-P2 — das fernöstliche Wunder	159	12/85
	DMPG9 — eine solide Sache	162	12/85
	Das Doppelleben des Joystick-Ports: 10er-Tastaturen	50	09/85
	Joysticks: Test und Marktübersicht (+Fehlerteufel 12/85)	19	11/85
	Es geht auch anders: Lightpens und Trackballs	22	11/85
EPROMer	Frisch gebrannt ist halb gespeichert (EPROM-Programmiergeräte im Test)	39	07/85
	QuickByte II — das Kraftpaket	14	10/85
Floppy/Datasette	Turbo-Floppies, zweite Generation: Speeddos plus + Prologic DOS	28	10/85
	Das große Rennen: Schnelle Bandlaufwerke	37	10/85
	Professionelle Floppylaufwerke für den C 64 (IEC-Floppies)	30	10/85
	Gut gekauft ist halb gespeichert (Marktübersicht Disketten)	38	10/85
Grafik	Die Videowerkstatt (Digitizer-Test)	32	05/85
	Digitalbilder m.d. C 64: PrintTechnik Digitizer	24	01/85
Interface	Hardware-Interface ganz weich: Test EC 64	23	01/85
	Gute Connections — Übersicht Schnittstellen	21	03/85
	Card/Print +6 — Das Allround-Interface	20	03/85
	Das Wiesemann-Centronics-Interface	18	03/85
	Erst ein IEC-Bus öffnet Tür und Tor (+Fehlert.4/6-85)	24	03/85
Monitore	Marktübersicht: Monochrome Monitore	30	12/85
Musik	Trommelwirbel: Test Digital Drums	45	08/85
	Die Musikhardware zum C 64	17	09/85
Roboter	Roboter selbst gebaut (Fischertechnik)	167	10/85
Scanner	So lernt Ihr Drucker lesen	30	06/85
Speicher	Speichertuning VC 20: Test 64 KByte Karte	26	01/85
Steuern	Flottes Türmchen: MEA-Interface	116	08/85

Kurse

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler ist keine Alchimie, Teil 5	142	01/85
	Assembler ist keine Alchimie, Teil 7	124	03/85
	Assembler ist keine Alchimie, Teil 9	138	05/85
	Assembler ist keine Alchimie, Teil 10	127	07/85
	Assembler ist keine Alchimie, Teil 11	126	08/85
	Assembler ist keine Alchimie, Teil 12	109	09/85
	Assembler ist keine Alchimie, Teil 13 (Schluß)	143	10/85
C 128	Entdeckungsreise duch den C 128	42	12/85
Effektives Programieren	Müllabfuhr im Computer: Garbage Collection, Teil 1	122	01/85
	Finden mit System, eine neuartige Suchmethode, Teil 3	148	03/85
	Sortieren mit dem Computer, Teil 2	159	05/85
	Sortieren mit dem Computer, Teil 3	124	06/85
	Sortieren mit dem Computer, Teil 4	138	08/85
	Sortieren mit dem Computer, Teil 5	124	09/85
	Sortieren mit dem Computer, Teil 6 (Schluß)	150	12/85
Extern	C 64 extern — Der Weg nach draußen, Teil 1	144	08/85
	C 64 extern — Der Weg nach draußen, Teil 2	122	09/85
	C 64 extern — Der Weg nach draußen, Teil 3 (Schluß)	129	10/85
Floppy	In die Geheimnisse der Floppy eingetaucht, Teil 4	148	01/85
	In die Geheimnise der Floppy eingetaucht, Teil 5	130	03/85
	In die Geheimnise der Floppy eingetaucht, Teil 6	145	05/85
	In die Geheimnisse der Floppy eingetaucht, Teil 7 (Schluß)	116	06/85
	Directory-Manipulationen I	140	06/85
Floppy	Directory-Manipulationen II	163	10/85
Grafik	Hires 3 — 15 neue Basic-Befehle, Teil 2	136	03/85
	Hires 3 — Grafikkurs-Anwendung, Teil 3 (Schluß)	152	08/85
	Sprites ohne Geheimnisse	40	08/85
	Streifzüge durch die Grafikwelt, Teil 1	106	09/85
	Streifzüge durch die Grafikwelt, Teil 2	149	11/85
Logeleien	Logeleien, Teil 1	143	07/85
	Logeleien, Teil 2	136	08/85
	Logeleien, Teil 3 (Schluß)	115	09/85
Musik	Dem Klang auf der Spur, Teil 2	136	01/85
	Dem Klang auf der Spur, Teil 4	131	04/85
	Dem Klang auf der Spur, Teil 5	152	05/85
	Dem Klang auf der Spur, Teil 7	132	07/85
	Dem Klang auf der Spur, Teil 8	133	08/85
	Dem Klang auf der Spur, Teil 9	126	10/85
	Dem Klang auf der Spur, Teil 10 (Schluß)	157	11/85
Speicher	Memory Map mit Wandervorschlägen, Teil 3	126	01/85
	Memory Map mit Wandervorschlägen, Teil 5	144	03/85
	Memory Map mit Wandervorschlägen, Teil 7	120	06/85
	Memory Map mit Wandervorschlägen, Teil 8	140	07/85
	Memory Map mit Wandervorschlägen, Teil 9	129	08/85
	Memory Map mit Wandervorschlägen, Teil 10	112	09/85
	Memory Map mit Wandervorschlägen, Teil 11	133	10/85
	Memory Map mit Wandervorschlägen, Teil 12	145	11/85
	Memory Map mit Wandervorschlägen, Teil 13	146	12/85
Sprachen	Basic ist out — es lebe Forth	43	01/85
VC 20	Der gläserne VC 20, Teil 4	130	01/85
	Der gläserne VC 20, Teil 6 (Schluß)	155	03/85

Software-Tips

Stichwort	Titel	Seite	Ausgabe
C 128	Erste Fragen und Antworten zum C 128	14	09/85
	Fragen und Antworten zum 128er	20	10/85
	Fragen und Antworten zum 128er	40	12/85
Drucker	Der MPS 802 lernt Deutsch	30	05/85
	Centronics-Interface für jeden Bedarf	78	07/85
Textverarbeitung	Software Corner — professionelle Programme richtig eingesetzt (Vizawrite-Tips)	174	12/85
Tips & Tricks	Autoboot beim C 64	86	03/85
	Verbindungsfreundlich (Parallelschnittstelle des VC 20)	91	03/85
	Undefinierte Opcodes des 6502	84	03/85
	Durch POKEs zum Erfolg (Spiele-POKEs)	83	03/85
	Tips und Erweiterungen zu Hi-Eddi und Simons Basic	88	03/85
	Basic-Befehle im Griff	79	05/85
	Durch POKEs zum Erfolg: Spiele-POKEs	78	06/85
	Formatierte Eingabe	148	06/85
	Hi-Text (Text in Hires)	70	08/85
	Verbotene Variablen	66	09/85
	Verschiedene Routinen für Anfänger und Profis (+Fehlerteufel 12/85)	88	11/85
	Der Trick mit dem Joystick (Joystickabfrage)	24	11/85
	Verschiedene Tips für Anfänger und Fortgeschrittene	106	12/85

Software-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler? Assembler! (Einführung)	32	01/85
	Assembler-Bedienung leicht gemacht, Teil 1	169	12/85
DFÜ	Der erste Kontakt mit DFÜ	40	06/85
	Die Netze der Post: Btx, Datex-P, Telebox	46	06/85
	DFÜ — Was ist das?	44	06/85
	Mailbox für Anfänger	30	07/85

Stichwort	Titel	Seite	Ausgabe
Datei	Die wichtigsten Begriffe der Dateiverwaltung	42	05/85
	Dateiverwaltung ist nicht gleich Datenbank	44	05/85
	Dateiverwaltung: Was Sie beim Kauf beachten sollten	40	05/85
Drucker	Hardcopy leicht gemacht (wie programmiert man Hardcopies)	34	09/85
EPROM	Wie sage ich es meinem EPROM? (EPROM-Grundlagen)	35	07/85
Funktionen	Funktionen für Anfänger	164	05/85
Lernen	Besser lernen mit dem Computer	166	10/85
Musik	Klangprogrammierung ohne Ballast	19	09/85
Spiele	Taktik- und Strategiespiele	46	03/85
	Play by Mail und Play by Modem	153	09/85
Sprachen	Sprachen für Computer, Teil 2	46	05/85
Textverarbeitung	Von der Schreibmaschine zum Textsystem	34	03/85

Listings zum Abtippen

Stichwort	Titel	Seite	Ausgabe
Anwendung	Der C 64 als Handballtrainer (AdM)	52	01/85
	Ligatab — ohne Organisation kein Tor (LdM)	50	03/85
	Gut Ziel mit dem C 64 — Schützenvereinsergebnisse (AdM)	52	03/85
	Weißt du, wieviel Sternlein stehen (Sternkarte) (AdM) (+Fehlert. 6/85)	52	05/85
	Haushaltsbuchführung (AdM)	52	07/85
	Netzwerkanalyse: Ein Programm für Hobby-elektroniker (AdM)	52	08/85
	Prüfungsfragen (AdM)	52	09/85
	Fit in Latein mit dem C 64 (AdM)	52	10/85
	Lyrik-Maschine (AdM)	52	11/85
	Hypra-Platos (LdM)	50	11/85
	Der Chemie-Assistent (AdM)	52	12/85
	SMON Teil 3: Ohne gutes Werkz. geht es nicht	69	01/85
	Hypra-Ass (LdM)	51	07/85
	Neues vom SMON (+Fehlerteufel 11/85)	87	10/85
	Reassembler zu Hypra-Ass (+Fehlerteufel 12/85)	97	11/85
	Ergänzungen zu Hypra-Ass (bedingte Verzweigungen)	96	11/85
	Tips & Tricks zum SMON (inklusive Diskmonitor)	100	12/85
Bildschirmseite	Auflösung Wettbewerb Bildschirmseite: Drei Top-Programme	158	09/85
DFÜ	Terminalprogramm der Spitzenklasse (+Fehlerteufel 10/85)	149	07/85
Datei	SMU — Der Maskengenerator (LdM)	50	12/85
Drucker	Hi-Eddi-Druckerroutinen	69	06/85
	C 64 Schreiberling — Drucken wie gemalt	54	10/85
	Koalabilder Farbhardcopy auf Epson JX-80	39	11/85
Einzeiler	Die nächsten 14 aus d. Einzeilerwettbewerb	157	01/85
Floppy	Hypra-Load mal 4 (+Fehlerteufel 3/85)	82	01/85
	Diskettenmonitor	83	08/85
	Disk-Designer	70	09/85
	Herzoperation (Hypra-Load + Hypra-Ass + DOS5.1 + Centronics)	104	11/85
Grafik	Vier Pseudo-VICs mit 32 Sprites	76	01/85
	Hi-Eddi: Zeichen- und Malproggramm (LdM)	50	01/85
	Elektrotechnisches Zeichnen mit dem VC 20	71	03/85
	Mini-Grafik VC 20, Grafikhilfe	69	05/85
	Trickfilm mit dem C 64: Bewegte 3D-Grafik (LdM) (+Fehlerteufel 6/85)	51	05/85
	Kurvenplotten mit Hardcopy auf dem C 16	68	06/85
	Doppelte Grafikauflösung für C 128	33	11/85
	Bilder aus einer anderen Dimension (Apfelmännchen)	80	11/85
Intelligenz	VIC — das intelligente Programm (Wettbewerbssieger)	173	05/85
Musik	Sound Machine (+Fehlerteufel 10/85)	23	09/85
	Sound Master (Basic-Erweiterung)	31	09/85
Spiele	6510 — Die Suche nach der Prozessor	70	05/85
	Samurai (Strategiespiel)	72	06/85
	Schach dem C 64: Schachprogramm zum Abtippen	72	08/85
	Spielen auf zwei Bildschirmen: Zeichensatzscrolling (LdM)	51	09/85
	Pac-Man unter der Lupe	76	10/85
	Block Out	84	11/85
	Seekrieg per Telefon (Schiffe versenken per Modem)	82	12/85
Spielehilfe	Die Scroll-Maschine — D. Fenster zur Spielewelt (LdM) (+Fehlert. 11/85)	52	06/85
Sprachen	Tiny Forth Compiler (LdM) (+Fehlert. 9/85)	51	08/85
Textverarbeitung	Hypra-Text (LdM) (+Fehlerteufel 11/85)	50	10/85
	Drucksache — Hypra-Text, Teil 2	71	11/85
Tips & Tricks	Große Buchstaben	89	01/85
	Restore für Unterprogramme	90	01/85
Tips & Tricks	Parameterübergabe an Maschinenspracheprogramme	88	01/85
	Cursorsteuerung leicht gemacht	86	02/85
	22 Read Error — Theorie und Praxis	41	03/85
	Floppy-Lister (+Fehlerteufel 4/85)	82	03/85
	Longscreen beim VC 20	83	05/85
	C 16: Help und Trace verbessert	84	05/85
	Ordnung ist das halbe Leben (Directory-Sorter)	77	05/85
	Dokumentationshilfe, Cross-Referenz-Liste C 64 (Wettbewerb)	155	06/85
	Prost mit dem C 64: Gerätesteuerung über Userport (+Fehlerteufel 9/85)	76	06/85
	Fenster-Befehle für den C 16	84	07/85
	Elektronische Merkzettel	83	07/85
	File-Compactor	82	07/85
	REM-Killer (+Fehlerteufel 9/85)	75	07/85
	Basic-Start-Generator	74	07/85
	Komfortable Ein-/Ausgaberoutine	77	07/85
	Bildschirmmasken leicht erstellt	86	08/85
	Der Bitmap-Compander (HiRes-Bilder komprimieren)	81	08/85
	Hypra-Save	79	08/85
	'Procedure' — oder der C 64 kann lernen	78	08/85
	Aufgewickelt — Listingscrolling für VC 20	63	09/85
	Programmgenerator für den C 64	86	10/85
	Cross-Ref optimiert	83	10/85
	Spieletrainer: Spritekill	86	11/85
	Tipp-Utility	99	12/85
	Der EPROM-Automat (wie man Module macht)	90	12/85
	80-Zeichen-Grafik für den C 128	78	12/85
	Hyper Screen (Sprites auf dem Bildschirmrand)	76	12/85
Transfer	Der C 64 als PET: PET-Simulator	87	01/85
Unterprogramme	Formatierte Eingabe	156	01/85

Software-Tests

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler im Test Teil 1	34	01/85
Basic-Erweiterung	GBasic — Alles drin	28	01/85
	Macro-Basic: Die Unterprogramm-Bibliothek	137	06/85
	Darf es etwas mehr sein? — Test Business-Basic	120	08/85
	Das Intellectool	138	09/85
	Formel 64: Das Multitalent	158	12/85
DFÜ	Terminalprogramme: Übersicht	42	06/85
Datei	Vergleichstest — 7 Dateiverwaltungen auf einen Blick	118	07/85
	Aufgeräumt mit Mainfile II	157	10/85
Grafik	Malen auf dem Bildschirm (Malprogramme)	34	08/85
	Grafikprogramme auf einen Blick: Marktübersicht	38	08/85
	Vergleichstest: Grafik-Erweiterungen	37	09/85
Lernen	Softlearning — die weiche Welle des Lernens	40	01/85
	Vokabeltraining mit dem Computer	39	03/85
	Marktübersicht: Lernsoftware	168	10/85
Musik	Musik für den C 64: Übersicht Musiksoftware	26	09/85
	The Music System — Zwei auf einen Schlag	164	12/85
Sprachen	Logo — die Sprache für Einsteiger	135	05/85
	Der Ada Trainingskurs auf dem C 64	129	05/85
	Promal — die neue Sprache für Profis?	124	07/85
	Forth-wärts mit M&T-Forth 64	126	07/85
	Was leistet Pilot?	121	08/85
	Pascal für Profis (Profi-Pascal)	122	08/85
	Super-Forth 64	144	09/85
	C — die professionelle Programmiersprache für den C 64	140	09/85
	Basic 7.0 — Das Superbasic des C 128	18	10/85
	Comal 80 — die universelle Programmiersprache	151	10/85
	Turbo-Pascal auf dem C 128	30	11/85

Stichwort	Titel	Seite	Ausgabe
Textverarbeitung	Homeword - Textverarbeitung zu Hause	36	03/85
	Totl-Text — Flexibilität ist Trumpf	38	03/85
	Protext — Textprofi mit 80 Zeichen	133	05/85
	Textomat Plus kontra Vizawrite	132	06/85
	Der Preishammer (Test: StarTexter)	135	09/85
	Paperclip — ausdrücklich gut	44	11/85

So machen's andere

Stichwort	Titel	Seite	Ausgabe
Semmeln	Semmelservice mit dem C 64	147	06/85
Sport	Commodore Sportservice: Heimcomputer zur Turnierauswertung	157	07/85
Hilfe	Computer für Behinderte	182	12/85

ASCII-Codes über $7F einbauen

Nachdem wir schon im letzten Abschnitt gesehen haben, was sich mit den ASCII-Codes kleiner als $80 alles anfangen läßt, wollen wir endlich auch die ASCII-Codes größer $7F einsetzen.

Zunächst eine kurze Aufstellung der für uns verwendbaren Steuerzeichen, deren ASCII-Codes über $7F liegen:

Steuerzeichen		ASCII-Code
orange $81	=	#129
Unterstreichen aus $82	=	#130
<SHIFT+RETURN> $8D	=	#141
Groß-/Grafik-Schrift ein $8E	=	#142
Blinken aus $8F	=	#143
schwarz $90	=	#144
CRSR UP	=	Cursor rauf
$91	=	#145
RVS OFF	=	Invers aus
$92	=	#146
CLR	=	Bildschirm löschen
$93	=	#147
INS	=	Zeichen einfügen
$94	=	#148
braun $95	=	#149
hellrot $96	=	#150
grau 1 $97	=	#151
grau 2 $98	=	#152
hellgrün $99	=	#153
hellblau $9A	=	#154
grau 3 $9B	=	#155
purpur $9C	=	#156
CRSR LEFT	=	Cursor links
$9D	=	#157
gelb $9E	=	#158
türkis $9F	=	#159

Laden Sie nun noch einmal das Programm Versuchskaninchen, und wir versuchen, den ASCII-Code für »Groß-/Grafik-Schrift ein« zu integrieren:

>01C00 00 24 1C 64 00 8F 8E 22 50...

oder >1C06 8E eingeben. Natürlich hat der Monitor den ASCII-Code korrekt untergebracht, aber beim LISTen wird nicht auf Groß-/Grafik-Schrift umgeschaltet, sondern stattdessen erscheint der Basic-Befehl RETURN, der als Token den Wert $8E hat.

Unser Problem ist also, daß der Interpreter beim LISTen unsere Steuerzeichen, sofern diese größer als $7F sind, für Token hält und somit decodiert - daher der Klartext RETURN.

Wenn wir unsere Codes in Anführungszeichen setzen, erscheint aber ein inverses Zeichen als Symbol für das Steuerzeichen. Auch damit können wir nicht zufrieden sein. Dennoch gibt es einen Ausweg, der mit viel Tüftelei entwickelt wurde.

Der »quote mode« hilft weiter

Der Interpreter und die Ausgaberoutine des Betriebssystems haben beide einen »quote mode« (Anführungszeichen-Modus), wobei der Interpreter die Zeichen unverändert ans Betriebssystem übergibt - auch Werte größer $7F, sofern sie in Anführungszeichen stehen.

Die einzige Lücke ist die, daß beim Betriebssystem der »quote mode« durch das Steuerzeichen CR (auch <SHIFT+RETURN>) verlassen wird, während der Anführungszeichen-Modus beim Interpreter lediglich von der Anzahl der offenen Anführungszeichen abhängt.

Dies machen wir uns mit Hilfe der Steuerzeichensequenz $22 0D zunutze. Wenn diese beiden Zeichen - in der genannten Reihenfolge - ausgegeben werden, so stellt das Byte $22 sowohl das Betriebssystem als auch den Basic-Interpreter in den Anführungszeichen-Modus. Das $0D-Byte bewirkt dann, daß das Betriebssystem den »quote mode« verläßt und somit alle Steuerzeichen, die seine Ausgaberoutine erhält, tatsächlich ausgibt (und nicht nur ein inverses Zeichen als Symbol für ein Steuerzeichen); die LIST-Routine des Basic-Interpreters kümmert sich um das $0D-Byte jedoch nicht und wähnt sich nach wie vor im Anführungszeichen-Modus.

Nun geschieht folgendes, wenn ein ASCII-Code wie $93 ausgegeben wird:

- Der Interpreter ist im »quote mode« und führt keine Decodierung durch, sondern sendet das Zeichen unverändert an die Ausgaberoutine des Betriebssystems. Würde dieses auch im »quote mode« stehen - wovon die LIST-Routine fälschlicherweise ausgeht -, so würde ein eventuelles Steuerzeichen (wie $93 = CLR = Bildschirm-Löschen) nicht ausgeführt, sondern durch ein inverses Zeichen dargestellt (bei $93 ein inverses Herz).

Beispiele

- Die Ausgaberoutine des Betriebssystems erhält das Steuerzeichen und führt dieses aus, da sie sich seit dem $0D-Byte (CR) im Normal-Modus befindet. Dies hat der Interpreter aber nicht feststellen können.

Zugegeben, der Trick ist ziemlich schwer zu verstehen, aber dafür ist die Anwendung wirklich kein Problem. Halten wir einfach die Wirkung fest:

Nach der Sequenz $22 0D werden alle Steuerzeichen, die darauf folgen, ausgeführt und weder als Token aufgefaßt (was eine Umwandlung in den Klartext zur Folge hätte), noch erscheint ein inverses Zeichen als Symbol für das Steuerzeichen.

Probieren wir es aus:

>01C38 1C 78 00 8F 22 0D 93 2A 2A...

oder einfach >1C3C 22 0D 93 eingeben.

$93 ist der ASCII-Code für CLR/HOME (Bildschirm löschen). Nun wird beim LISTen von Zeile 120 der Bildschirm gelöscht. Wenn wir aber auch nur einen Teil der Sequenz 22 0D weglassen, so erscheint entweder der Basic-Befehl LOAD (wenn das 22 fehlt) oder ein inverses Herz (wenn das 0D fehlt, aber das 22 vorhanden ist).

Dies war ein Beispiel für die Anwendung der Sequenz 22 0D.

Wir wollen jedoch fortan die Sequenz 22 14 0D verwenden. Das eingefügte 14 ist der ASCII-Code für DEL (DELETE) und löscht unverzüglich das Anführungszeichen. In diesem Falle (Bildschirm löschen folgt unmittelbar danach) ist es zwar unwichtig, aber wenn nicht gerade ein solches Zeichen folgt, sollte man darauf achten, daß das Anführungszeichen verschwindet:

>01C38 1C 78 00 8F 22 14 0D 93 2A...

oder >1C3C 22 14 0D 93 eingeben.

Statt der Sequenz 22 14 0D ist auch 22 1B 44 0D möglich, wenn nicht nur das Anführungszeichen, sondern die gesamte Bildschirmzeile gelöscht werden soll.

Nun wollen wir aber außer dem 93 auch noch weitere Steuerzeichen einbauen, um ein wenig Übung zu bekommen, damit Sie auch andere Programme als unser Versuchskaninchen manipulieren können. Zwar haben wir alle möglichen Codes genannt, besprechen werden wir jedoch nur die wichtigsten.

Umschalten auf Groß-/Grafik-Schrift

Damit beim LISTen der Groß-/Grafik-Zeichensatz eingestellt wird, bauen wir das entsprechende Steuerzeichen ein:
>01C40 8E 2A 2A 2A...
oder >1C40 8E eingeben.

Nun wollen wir noch das Umschalten mittels <SHIFT+CBM> verhindern:
>01C40 8E 0B 2A 2A...
oder >1C41 0B eingeben. Daran können wir auch eindeutig erkennen, daß Steuerzeichen mit ASCII-Codes größer $80 weiterhin funktionsfähig bleiben (damit wirklich kein Mißverständnis auftritt).

Die restlichen Farben

Als uns die Sequenz 22 14 0D noch nicht bekannt war, konnten wir nur 4 von 16 Farben über ASCII-Codes einbauen. Hier sind die restlichen 12 Codes:

Farbe		ASCII-Code
orange $81	=	#129
schwarz $90	=	#144
braun $95	=	#149
hellrot $96	=	#150
grau 1 $97	=	#151
grau 2 $98	=	#152
hellgrün $99	=	#153
hellblau $9A	=	#154
grau 3 $9B	=	#155
purpur $9C	=	#156
gelb $9E	=	#158
türkis $9F	=	#159

Jeden dieser zwölf Codes auszuprobieren würde nichts bringen. Wir arbeiten mit einigen wenigen Beispielfarben, denn für eine andere Farbe ist nur ein anderer – obiger Tabelle zu entnehmender – Code zu verwenden.

Wir setzen jetzt den Code für gelb, nämlich $9E, ein:
>01C40 8E 0B 9E 2A 2A...
oder >1C42 9E.

An späterer Stelle setzen wir noch die Codes für braun und türkis ein, wo wir dann nicht mehr näher darauf eingehen müssen.

Ähnlich wie CR: SHIFT/RETURN

Nur exemplarisch wollen wir den Code für <SHIFT+RETURN> verwenden, denn der CR-Code $0D ist viel geeigneter (da er zwar dieselbe Wirkung bei Ausgaben hat, aber aufgrund der Tatsache, daß der ASCII-Code $0D kleiner als $80 ist, leichter eingesetzt werden kann):
>01C40 8E 0B 9E 8D 8D 2A 2A...
oder >1C43 8D 8D eingeben.

Wie gesagt: $8D (SHIFT/RETURN) geht nur nach der 22 14 0D-Sequenz, $0D (CR) arbeitet auch ohne diese und ist leichter zu handhaben.

Mit RVS OFF reverse und nicht-reverse Darstellung in einer Zeile

Während der ASCII-Code von RVS ON (inverse Darstellung ein) bisher verwendbar war, da er mit einem Wert von $12 kleiner als $80 ist, können wir erst aufgrund der Sequenz 22 14 0D auf den ASCII-Code von RVS OFF zurückgreifen.

Natürlich ist der RVS OFF-Code erst sinnvoll, falls er dann erfolgt, wenn gerade die reverse Darstellung eingeschaltet ist. Daher wollen wir bewirken, daß ein Stern revers und der nächste unmittelbar danach nicht-invers ausgegeben wird:
>01C44 8D 12 2A 92 2A 2A 2A...
oder einfach >1C45 12 und >1C47 92 eingeben.

$12 ist der RVS ON-, $92 = #146 der RVS OFF-Code.

Als kleine Erweiterung stellen wir nach der RVS-Spielerei die aktuelle Zeichenfarbe auf braun:
>01C44 8D 12 2A 92 2A 95 2A 2A...

Blinkende und nicht-blinkende Zeichen in einer Zeile

Bei den Codes unter $80 haben wir den FLASH ON-Code ausdrücklich erwähnt und eingebaut. Er hat den ASCII-Code $0F und bewirkt, daß die danach ausgegebenen Zeichen blinken – allerdings nur bei Verwendung des 80-Zeichen-Bildschirms.

Das Steuerzeichen zum Ausschalten dieses Blinkens bewirkt nicht, daß das Blinken aufhört, sondern nur, daß die danach ausgegebenen Zeichen nicht mehr blinken.

Wir wollen nun einen blinkenden Stern und einen darauffolgenden, der nicht blinkt, ausgeben lassen:
>01C48 2A 95 0F 2A 8F 2A 2A 2A...
oder >1C4A 0F 2A 8F 2A eingeben.

$8F ist der ASCII-Code von FLASH OFF.

Unterstrichene und nicht-unterstrichene Zeichen in einer Zeile

Ähnlich wie bei den blinkenden Zeichen ist es mit den ASCII-Codes $02 (Unterstreichen bei 80-Zeichen-Modus ein) und $82 (Unterstreichen aus, geht auch nur bei 80-Zeichen-Bildschirm).

Wir lassen nun einen unterstrichenen und einen nicht-unterstrichenen Stern »*« ausgeben:
>01C4D 2A 02 2A 82 2A 2A 2A...
oder >1C4E 02 2A 82 2A eingeben.

Außerdem ist wieder einmal ein Farbwechsel fällig:
>01C4D 2A 02 2A 82 2A 9F 2A...
$9F ist der ASCII-Code für türkis.

Damit haben wir einige interessante Manipulationen durchgeführt. Die Anwendung von diesen bleibt Ihrer Phantasie überlassen, denn sie ist so vielfältig, daß wir nur die wichtigsten Aspekte besprechen müssen. Außerdem ist das Prinzip immer wiederkehrend: anstelle eines Füllzeichens (Sternchen) wird ein ASCII-Code eingesetzt. Mit ein wenig Experimentieren kommt man ziemlich weit und kann seine Listings wirklich interessant gestalten.

Anwendung der ASCII-Codes auf Maschinenprogramme mit einer Basic-Zeile

Die ASCII-Codes im Listing sind natürlich vor allem für Basic-Programme geeignet. Aber auch bei Maschinenprogrammen, die eine Basic-Zeile mit einem SYS-Befehl haben, damit sie über den Basic-Befehl RUN gestartet werden können, kann man mit den ASCII-Codes einiges erreichen.

So kommen unter anderem folgende Anwendungen, die wir auch besprechen wollen, in Frage:

- Kommentar wie Copyright, Hinweise oder ähnliches ausgeben

- SYS-Befehl nach LISTen wieder löschen
- Cursor auf RUN positionieren, damit der Start bequemer ist

Als Beispielzeile für unsere Manipulationen verwenden wir

2000 BANK 15:SYS DEC("41BA"):REM "**********"

In der REM-Zeile stehen übrigens 10 Sternchen.

An der über diese Zeile angesprungenen Adresse steht ein RTS-Befehl, Sie müssen also nicht fürchten, daß ein versehentlicher Programmstart zum Absturz führt.

Als erstes wollen wir, daß anstelle der Zeile nur der Hinweis »(C) 128« ausgegeben wird. Dazu suchen wir zunächst das Anführungszeichen und die Sternchen über

H 1C01 5000 '"**********

und erhalten als Wert 01C19.

Jetzt speichern Sie bitte unsere exemplarische Basic-Sys-Zeile, denn wir wollen eine kleine Manipulation durchführen, die zunächst das Listing der Zeile löscht und dann den Text »(C) 128« ausgibt:

>01C19 1B 44 22 28 43 29 31 32 38 22 07 0A

Außerdem wird ein Klingelzeichen erzeugt und eine Leerzeile eingefügt (durch LF). 1B 44 ist ESC-D, dann kommt der Text »(C) 128« und schließlich 07 (das Klingelzeichen) und 0A (LF).

Als nächstes wollen wir noch, daß beim LISTen anstelle der Zeile der RUN-Befehl erscheint und auf diesem nach dem Listen der Cursor steht, damit der Anwender nur noch auf <RETURN> drücken muß, um das Programm zu starten. Dazu löschen wir die Zeile, geben zweimal CR aus, dann den Text RUN und fahren schließlich den Cursor dreimal nach oben, damit auch nach der Ausgabe von »READY.« und einem CR nach dem Listen der Cursor auf dem RUN steht. Da wir den CRSR UP-Code brauchen, muß zudem irgendwo die Sequenz 22 14 0D, die wir hier zum Löschen der ganzen Zeile durch 22 1B 44 0D ersetzen, stehen:

>01C19 22 1B 44 0D 0D 0D 52 55 4E 91 91 91

Probieren Sie jetzt einmal den LIST-Befehl aus.

Durch eine solche Änderung kann man es vor allem dem Anfänger leichtmachen.

Natürlich ist es auch – sofern man vorher genügend Füllzeichen eingegeben hat – möglich, andere Befehle als RUN vorzubereiten.

Damit wären wir am Ende unseres Kurses angekommen. Wir hoffen, es hat Ihnen Spaß gemacht und neue Möglichkeiten eröffnet. Ganz abgesehen davon, daß wir nun den Monitor wirklich beherrschen und für Basic- und Maschinenprogramme bestens einsetzen können, haben wir uns auch nebenbei elementare Kenntnisse angeeignet: Sie können nun mit den wichtigsten Zahlensystemen umgehen, wissen über die Speicherstruktur Bescheid und kennen den Aufbau eines Basic-Programms im Speicher.

Wenn Sie jetzt andere Literatur lesen und vor allem verstehen wollen, so fällt es Ihnen auf jeden Fall leichter. Auch wenn Sie vorher nicht mit Maschinensprache weit gekommen sind, sollten Sie es jetzt noch einmal versuchen; mittlerweile geht es sicherlich viel besser.

(F. Müller/ks)



Wunderwelt der Grafik

Das Basic des C128 verfügt über wunderbare Befehle zur Nutzung der HiRes-Grafik. Wir zeigen Ihnen Anwendung und Programmierung dieser HiRes-Grafik.

Hier geht es um die hochauflösende und die Mehrfarbengrafik, die wir uns im Detail ansehen werden. 14 Befehle steuern im Basic 7.0 des C128 die Grafikmöglichkeiten im engeren Sinn. Zählt man noch die Sprite- und Shapeanweisungen hinzu, dann verfügen wir über 27 grafische Werkzeuge in Basic. Wir erklären Ihnen die Anwendung und Programmierung jedes einzelnen dieser 14 Befehle, die Ihnen das Arbeiten mit der hochauflösenden Grafik erleichtern.

GRAPHIC

Dieser Befehl dient - der Name sagt es schon - zum Umschalten in die verschiedenen Grafik- und Textmodi unseres Computers. Mittels

```
GRAPHIC A, B, C
```

können je nach den Kennungen A, B und C eine ganze Reihe von Optionen angewählt werden.

Die Kennung A aktiviert den Grafik-Modus:

A=0 Der Textbildschirm - und zwar derjenige mit 40 Zeichen Zeilenbreite - wird eingeschaltet

A=1 Die hochauflösende Grafik wird aktiviert

A=2 Ein »Splitscreen«, also ein Bildschirm, auf dem sowohl 40-Zeichen-Text als auch hochauflösende Grafik zulässig sind, kann damit eingeschaltet werden

A=3 Die Mehrfarbengrafik (auch Multicolorgrafik genannt) ist angewählt

A=4 Auch dieser Parameter erlaubt die Einrichtung eines Splitscreen. Diesmal erscheint zum 40-Zeichen-Text-Schirm noch die Mehrfarben-Grafik

A=5 Schaltet auf 80 Zeichen um.

Der Parameter B entscheidet darüber, ob der ausgewählte Bildschirm gelöscht wird oder nicht:

B=0 Es wird nicht gelöscht

B=1 Der Bildschirm (in den Grafik-Modi 1 bis 4 auch die Bit-Map genannt) wird gelöscht.

Der Parameter C spielt lediglich in den Modi 2 und 4 eine Rolle (Splitscreen). Er bestimmt, an welcher Zeile der Textbildschirm eingeblendet werden soll. Die Splitscreens sind so eingerichtet, daß immer von dieser Zeile an abwärts Textbereich existiert. Voreingestellt - und daher muß C nicht immer angegeben werden - ist die Zeile 19.

Worin unterscheiden sich diese Grafik-Modi? Wenn Sie im C64-Modus Grafik programmieren wollen, dann kommen Sie ohne einen sehr tiefen Einblick in die Speicherorganisation des Computers nicht sehr weit. Auch wenn uns das der C128 durch sein leistungsfähigeres Basic erspart: Ein wenig sollten Sie dennoch wissen.

Die Grafik-Modi

40-Zeichen-Text

Zur Darstellung des 40-Zeichen-Bildschirms belegt der Computer zwei Speicherbereiche:

1024 bis 2023 Bildschirm-RAM und
55296 bis 56295 Bildschirm-Farben-RAM

In jeweils 1000 Speicherstellen (40 Spalten mal 25 Zeilen) befindet sich die Text-Information, die auf dem Monitor oder Fernseher zu sehen ist. Die Aufteilung dieser beiden Speicherbereiche finden Sie in Ihrem Handbuch auf den Seiten 5-80 und 5-82. Wenn Sie dafür sorgen, daß in der linken oberen Ecke Ihres Bildschirms ein A steht, dann finden Sie durch

```
BANK 0:PRINT PEEK(1024)
```

die Zahl 1 in der Speicherstelle 1024. Das ist der Bildschirm-Code des Zeichens »A«. Umgekehrt können Sie durch POKEn auch Zeichen in den Bildschirmspeicher schreiben:

```
BANK 0:POKE 1025,2
```

schreibt ein »B« an die zweite Position der ersten Zeile. Im Handbuch sind die Bildschirm-Codes der Zeichen zu finden. Der andere Speicherbereich ab 55296 versieht die Zeichen mit Farbe. In den alten Versionen des C64 mußten zwei POKEs eingegeben werden, um ein Zeichen sichtbar werden zu lassen. Der zweite POKE diente zum Einschreiben eines Farb-Codes in die dazugehörige Bildschirmfarbspeicherstelle. Hier ist das nicht nötig: Der Speicherbereich ab 55296 wird automatisch mit entsprechenden Werten gefüllt: Die unteren 4 Bit enthalten nach dem Einschalten die Zahl 13, was dem Code der Farbe hellgrün (vermindert um 1) entspricht. Es steht Ihnen aber frei, bunte Buchstaben zu erzeugen. Dazu ist lediglich ein anderer Farb-Code in die zum Bildschirmspeicher gehörige Farbzelle zu schreiben. Bild 1 soll dieses Zusammenspiel beider Speicherbereiche verdeutlichen:

Übrigens, falls Ihnen die aktuelle Cursorfarbe nicht mehr gefallen sollte, dann können Sie sie durch


```
BANK 0:POKE 241,Farbcode
```

ändern. Auch alle danach geschriebenen Zeichen erhalten die so gewählte neue Farbe.

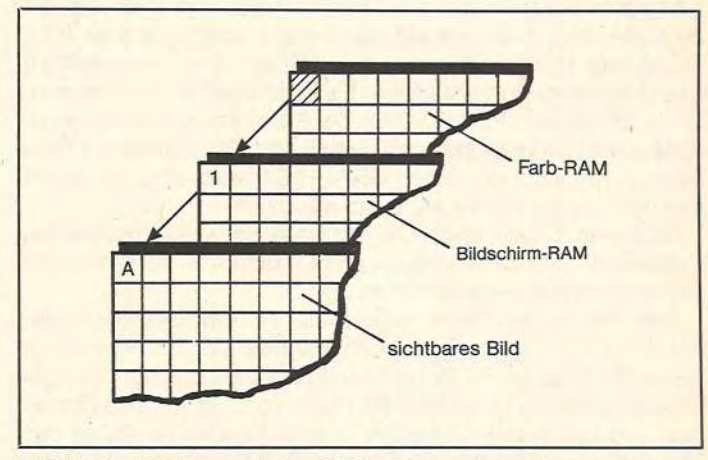


Bild 1. Ein Zeichen auf dem Bildschirm setzt sich aus der Bildschirm- und der Farbinformation zusammen

Nun wissen wir zwar, daß zum Darstellen eines Zeichens auf dem Bildschirm ein Zeichen-Code im Bildschirmspeicher und ein Farb-Code im dazugehörigen Farbspeicher nötig sind, so ganz befriedigend ist das als Erklärung aber noch nicht. Woher weiß unser Computer eigentlich, was er abbilden soll, wenn beispielsweise der Zeichen-Code 1 in der Speicherstelle 1024 steht? Für jedes Zeichen, das der Computer darstellen kann, gibt es im sogenannten Character-ROM ein Muster. Der Zeichen-Code dient als Zeiger auf das erste Byte eines solchen Musters. Die 1 deutet auf das erste Zeichenmuster, das zum großen Buchstaben »A« gehört. Es gibt übrigens auch das nullte Zeichen, den »Klammeraffen«.

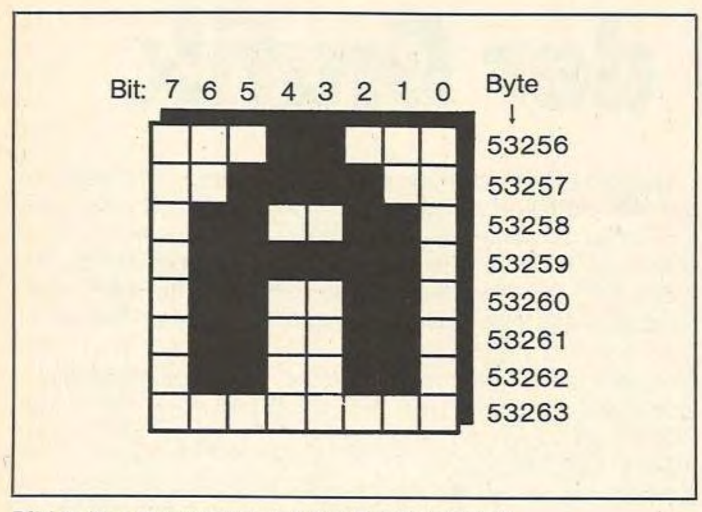


Bild 2. Das Zeichenmuster des Buchstaben A

Jedem Zeichen sind im Character-ROM 8 Byte zugeordnet. Ein Beispiel zeigt Bild 2.

Daraus ist zu ersehen, daß jedes Zeichen als gesetzte oder nicht gesetzte Punkte in einem 8x8-Rasterfeld definiert ist. Falls Sie über einen guten Monitor verfügen und Kontrast sowie Helligkeit entsprechend einstellen, dann können Sie diese Punkte auch auf dem Bildschirm erkennen.

Zwei Fragen stellen sich aus dieser Erkenntnis heraus:
- Kann man eigene Zeichen erstellen und verwenden?
- Gibt es eine Möglichkeit, statt jeweils auf solch ein 8x8-Raster auch auf einzelne Punkte einzuwirken?

Die erste Frage wird im Abschnitt über den VIC-Chip beantwortet. Hier sei nur erwähnt, daß es möglich ist.

Die zweite Frage führt uns zu den anderen Grafik-Modi.

Der Hochauflösungsmodus

Wir hatten festgestellt, daß wir im Textmodus (40 Zeichen) immer nur (beispielsweise durch POKE 1024,1) auf ein 8x8-Punkte-Raster zugreifen können. Die Auflösung unseres Bildschirms beträgt in diesem Zustand 40 x 25 Zeichen. Es gibt noch einige Tricks, eine sogenannte Viertelpunktgrafik zu entwerfen, in der die auf der Tastatur vorgegebenen Grafikzeichen (beispielsweise mit <CBM+C> zu erhalten) geschickt verwendet werden. Es juckt aber in den Fingern, jeden Bildpunkt eines solchen 8 x 8-Zeichens einzeln anzusprechen. Das ist tatsächlich mittels dem Grafikmodus 1 (und auch 2) möglich, der durch GRAPHIC1 anwählbar ist. Damit erhöht sich die Auflösung ganz erheblich:

320 horizontale und 200 vertikale Bildschirmpositionen stehen uns hier zur Verfügung. Wir haben den sogenannten Bit-Map-Modus eingeschaltet.

Das Prinzip ist dabei folgendes: Jedem Bildpunkt entspricht ein Bit eines Speicherbereiches, der Bit-Map. Ist in dieser Bit-Map ein Bit gleich 1, dann erscheint an der dazugehörigen Bildschirmposition ein Punkt (das ist wie das Eintragen von Geländeeinzelheiten in eine Landkarte, daher der Name Map = Landkarte). Jeweils 8 Bit ergeben ein Byte, woraus sich auf die Größe der Bit-Map schließen läßt: (320*200)/8 = 8000 Byte.

Alles, was auf dem Bildschirm erscheint, ist der Inhalt dieser Bit-Map, die durch den GRAPHIC1-Befehl eingerichtet wird. Wo befindet sie sich? Wenn Sie sich mal vor und nach dem Einrichten der Bit-Map durch PRINT FRE(0) den freien Speicherplatz ausgeben lassen, dann erkennen Sie folgendes: In der BANK 0, also dem Basic-Textspeicher, wird einiges verändert. Normalerweise (also ohne Bit-Map) beginnt der Basic-Text in Speicherstelle $1C00 (dezimal 7168). GRAPHIC1 schiebt den Textstart (und ein eventuell schon im Speicher befindliches Programm) in Windeseile bis $4000 (dezimal 16384). Die Bit-Map beginnt ab $2000 (dezimal 8192).

GRAPHIC1 legt aber nicht nur eine Bit-Map an und schaltet den Videochip auf den speziellen Grafikmodus um, sondern definiert auch ein Koordinatensystem auf dem Bildschirm, das sich aus der Bit-Anordnung des Bildes ergibt. Sehen Sie dazu das Bild 3.

Wir finden einen Koordinatenursprung in der linken oberen Bildecke, eine nach unten weisende Y-Achse und eine nach rechts verlaufende X-Achse: Ein solches linkshändiges System bereitet dem Anwender manchmal einige Probleme, weil in der Mathematik ein rechtshändiges System eingesetzt ist (da wäre dann der Ursprung unten links zu finden und die Y-Achse verliefe nach oben). Alle Zeichenbefehle beziehen sich im Normalfall auf die so festgelegten Koordinaten. Ein Befehl

```
DRAW 1,0,0 TO 319,199
```

(zu den Einzelheiten kommen wir noch) zeichnet also eine Linie von links oben nach rechts unten.

Wir haben noch nicht erwähnt, wie der Computer die Farben in diesem Modus festlegt. Das soll beim COLOR-Befehl behandelt werden. Auf Koordinatensysteme kommen wir beim SCALE-Befehl noch einmal zurück.

Die Mehrfarbengrafik

Während in den Grafik-Modi 1 und 2 jedes gesetzte Bit in der Bit-Map einen Bildpunkt nach sich zieht, ist die Mehrfarbengrafik (Modi 3 und 4) immer auf Bit-Paare orientiert. Die Bit-Map hat zwar als Karte des Bildschirminhaltes ihre Bedeutung beibehalten, aber sie wird vom VIC-Chip nun anders gelesen. Es existieren vier Möglichkeiten von Bitkombinationen zu je zwei Bit:

00 Hintergrund
01 Farbe 1
10 Farbe 2
11 Farbe 3

Auf welche Weise welche Farbe angesprochen wird, soll uns beim COLOR-Befehl weiter beschäftigen. Hier interessiert uns die Auswirkung auf die Auflösung.

Weil in der X-Richtung nun nur noch Bit-Paare zählen, halbiert sich die Auflösung. Das Koordinatensystem der Multicolorgrafik ist ebenso aufgebaut wie das der hochauflösenden Grafik, lediglich in X-Richtung stehen uns nur noch 160 Positionen zur Verfügung. Bild 4 zeigt uns dieses neue Koordinatensystem.

Die Splitscreens

In den Grafik-Modi 2 und 4 wird der Bildschirminhalt durch jeweils zwei verschiedene Betriebsarten des VIC-Chip gewonnen. Im oberen Teil sehen wir den Inhalt der Bit-Map, im unteren aber 40-Zeichen-Text. Dies wird erreicht durch die sogenannte Rasterzeilenunterbrechung. Auch im C64-Modus kann mittels eines Maschinenprogramms diese Aufteilung des Bildschirms programmiert werden. Dabei gab es immer ein Problem, das mit der Bearbeitungszeit zusammen-

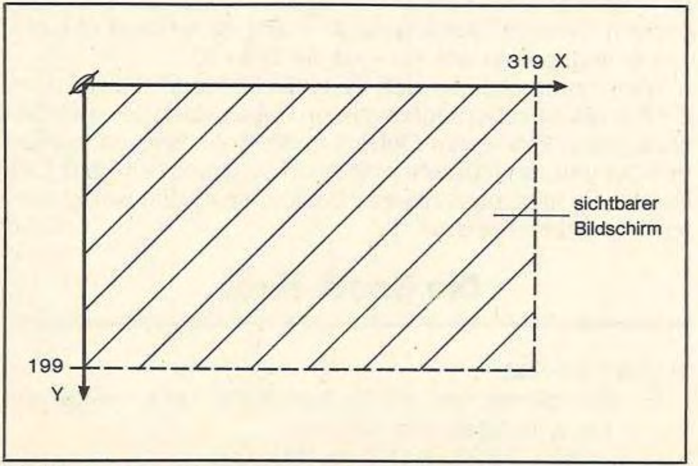


Bild 3. Das Koordinatensystem im HiRes-Modus

hing. Die Rasterzeilenunterbrechung und die normale Unterbrechung (die 60mal pro Sekunde stattfindet) durften sich nicht gegenseitig in die Quere kommen. Diese Schwierigkeit konnte im Programmbetrieb meistens ganz gut gelöst werden. Im Direktmodus führten aber Cursorbewegungen und die Arbeit mit Sprites oft zu einer zittrigen Grenze der beiden Darstellungsmodi auf dem Bildschirm. Dasselbe Problem scheint im C128-Modus dafür verantwortlich zu sein, daß diese Grenze manchmal geradezu extrem gestört wird, beispielsweise beim Steuern von Sprites über die Splitscreengrenze im Grafikmodus 4.

Beim Einschalten eines Splitscreen befindet sich der Textcursor nicht automatisch im Textbereich. Er muß erst dorthin gefahren werden. Zwei Möglichkeiten bieten sich hier an: Zum einen kann ein PRINT AT simuliert werden durch:

```
BANK 15:SYS 65520,,Zeile,Spalte
```

Zum anderen bietet sich die Definition eines Bildschirmfensters unter Benutzung des WINDOW-Befehls an. Man legt dann einfach den gesamten Textbereich als WINDOW fest. Ist also keine spezielle Trennzeile angegeben worden (dann tritt der Übergang von Grafik zu Text in Zeile 19 ein), kann das durch

```
WINDOW 0,20,39,24
```

geschehen. Übrigens bleibt das Bildschirmfenster auch nach einer Rückkehr in den Textmodus erhalten. Wird das nicht gewünscht, kann durch

```
WINDOW 0,0,39,24
```

der gesamte Bildschirm wieder benutzt werden.

Der 80-Zeichen-Textmodus

Sollten Sie stolzer Besitzer eines 80-Zeichen-Monitors sein, dann können Sie durch GRAPHIC5 die Textausgaben auf dessen Bildschirm lenken. Interessanterweise ist es nun möglich, zwei getrennte Anzeigen zu erhalten, beispielsweise die Multicolorgrafik auf dem 40-Zeichen-Schirm und die Textausgaben auf dem 80-Zeichen-Monitor. Die Splitscreen-Optionen braucht man dann nicht mehr – sie sind sogar hinderlich, weil sie die Ausgabe von Text wieder auf den 40-Zeichen-Schirm lenken und uns die Mühe machen, den verringerten Grafikausgabebereich zu berücksichtigen (die Y-Achse wird ja nun ab Zeile 19 überdeckt vom Textbereich).

Die Textdarstellung auf dem 80-Zeichen-Schirm folgt etwa dem gleichen Prinzip, wie wir es schon beim 40-Zeichen-Text kennengelernt haben – mit folgenden Unterschieden:

Zunächst benutzt unser Computer jetzt einen anderen Videobaustein, den VDC 8563. Wir werden diesen Videochip an anderer Stelle noch etwas genauer kennenlernen.

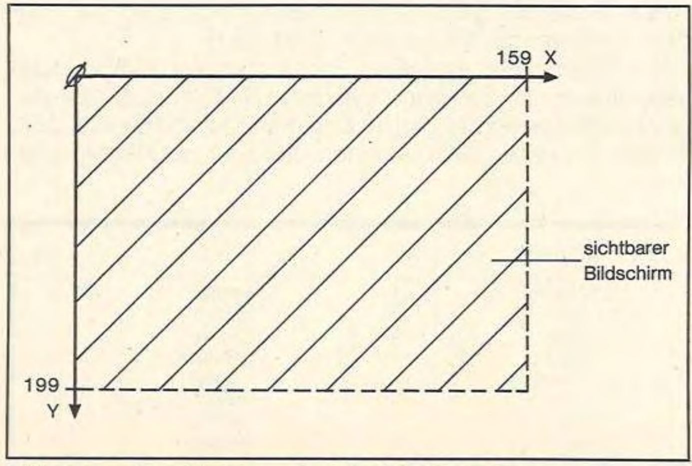
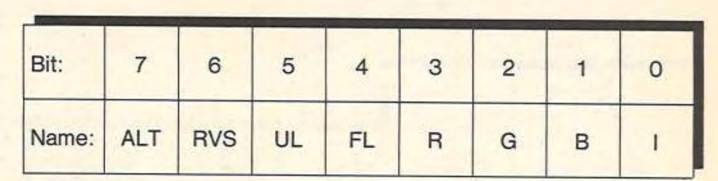


Bild 4. Das Koordinatensystem im Multicolor-Modus



Bit:	7	6	5	4	3	2	1	0
Name:	ALT	RVS	UL	FL	R	G	B	I

Bild 5. Aufbau eines Attribut-RAM-Byte

Der Bildschirmspeicher ist nirgendwo im Bereich der 128 KByte unseres Computers zu finden, die wir direkt ansprechen können (durch PEEK oder mittels Monitor). Ebensowenig werden wir einen Hinweis auf den Farbspeicher finden. Woran liegt das? Immerhin brauchen wir doch 80 x 25 Byte, also 2000 für das Bildschirm-RAM und ebenso viele Byte für die Farbzellen. Da sind wir auf eine Besonderheit des VDC-Chip gestoßen: Er benutzt einen eigenen 16-KByte-RAM-Bereich, der völlig getrennt vom normalen Speicher existiert und von uns nur über zwei Speicherstellen ansprechbar ist. Welche das sind und wie man sie benutzt, soll ebenfalls beim VDC besprochen werden. Hier nur die Aufteilung dieses Speichers:

0000 bis 2047 Bildschirm-RAM
2048 bis 4095 Attribut-RAM
4096 bis 8191 frei
8192 bis 16385 Zeichenspeicher

Zur Bedeutung des Bildschirm-RAM braucht nichts mehr gesagt zu werden. Das Attribut-RAM entspricht etwa dem Bildschirmfarbspeicher, birgt aber noch mehr Möglichkeiten in sich. Weil wir alle Aspekte der Farbgebung der anderen Modi beim COLOR-Befehl behandeln werden, mit diesem Befehl aber das VDC-Attribut-RAM nicht erschöpfend erfaßt werden kann, soll hier diese VDC-Spezialität erklärt werden. Zuvor noch ein paar Worte zum Zeichenspeicher. Während im 40-Zeichen-Textmodus ein Zeichenmuster aus dem Character-ROM entnommen wird, ist das im 80-Zeichen-Textmodus nicht der Fall. Vielmehr findet hier schon in der Einschaltphase ein Kopieren aller Zeichen in den speziellen VDC-RAM statt. Von dort stammen nun auch die Zeichenmuster. Sie kennen sicher schon die Erscheinung, daß beispielsweise beim Umschalten auf den DIN-Zeichensatz (mit der <CAPS/LOCK>-Taste) einige Zeit verstreicht, bis alle Zeichen im neuen Gewand zu sehen sind: Das liegt daran, daß der gesamte Zeichensatz umkopiert werden muß. Im Bereich 8192 bis 16385 finden sich danach nur noch die DIN-Zeichenmuster.

Es wird sicher Ihrer Aufmerksamkeit nicht entgangen sein, daß hier die Rede von RAM ist im Gegensatz zum Zeichen-ROM, das bei der 40-Zeichen-Textdarstellung eine Rolle spielt. Wenn wir wissen, wie man dieses VDC-RAM ansprechen kann, wird uns nichts daran hindern, auch in den Speicherbereich des Zeichensatzes hineinzuschreiben. So können dann eigene Zeichen definiert werden.

Kommen wir nun aber zurück zum Attribut-RAM. Jeder Bildschirmspeicherstelle entspricht ein Byte dieses Attributbereiches. Durch dieses Byte aber wird noch mehr gesteuert als die Farbe, was Sie am Aufbau solch einer Speicherstelle erkennen können (siehe Bild 5).

Die Bits 0 bis 3 bestimmen die Farbe, was wir gleich noch untersuchen wollen.

Bit 4: Ist hier eine 1 zu finden, dann blinkt das Zeichen.

Bit 5: Das Zeichen ist unterstrichen, wenn hier eine 1 enthalten ist.

Bit 6: Hiermit wird ein Aspekt der Zeichendarstellung »doppelt gemoppelt«: Die Revers-Darstellung. Eigentlich kann nämlich durch Einschreiben einer »1« an diese Stelle genau dasselbe erreicht werden, wie durch das ansonsten etwas speicherfressende Verfahren eines gesonderten Zeichenmusters, mit dem momentan die inversen Zeichen gezeichnet werden.

Bit: 3	2	1	0			
R	G	B	I	dez.	Farbe	entspricht Farbcode im 40-Z.-Modus
0	0	0	0	0	schwarz	1
0	0	0	1	1	d'grau	12
0	0	1	0	2	blau	7
0	0	1	1	3	h'blau	15
0	1	0	0	4	grün	6
0	1	0	1	5	h'grün	14
0	1	1	0	6	türkis	4
0	1	1	1	7	orange	9
1	0	0	0	8	rot	3
1	0	0	1	9	h'rot	11
1	0	1	0	10	lila	13
1	0	1	1	11	violett	5
1	1	0	0	12	braun	10
1	1	0	1	13	gelb	8
1	1	1	0	14	h'grau	16
1	1	1	1	15	weiß	2

Bild 6. Zuordnung der Bit-Kombinationen zu den Farben (stark vom verwendeten Monitor abhängig!)

Bit 7: Im Gegensatz zum 40-Zeichen-Textmodus, der die Groß- und Kleinschreibung nur durch Umschalten (per <CBM+SHIFT>-Taste) erlaubt, können im 80-Zeichen-Modus beide Zeichensätze gleichzeitig verwendet werden. Welches von den beiden möglichen Zeichen auf dem Bildschirm zu sehen ist, entscheidet dieses Bit. Liegt hier eine 1 vor, dann stammt das sichtbare Muster aus dem zweiten Zeichensatz.

Nun zu den Farb-Bits. Im Bild 5 sind diese mit R, G, B und I bezeichnet. Dabei steht R für Rot, G für Grün, B für Blau und schließlich I für Intensität. Je nach Mischung der Signale, die durch unterschiedliche Bitbelegungen der vier Bit erzeugt werden, ergeben sich die 15 möglichen Farbdarstellungen. Bild 6 zeigt alle Kombinationen.

All diese einzelnen Aspekte, die das Attribut-RAM erlaubt, können durch CHR$- oder/und ESC-Befehle angesteuert werden. So erzeugt

```
PRINT CHR$(2)"A"
```

ein unterstrichenes und

```
PRINT CHR$(27)+"F";"A"
```

ein blinkendes A. Das Handbuch gibt ab Seite 4-4 erschöpfend darüber Auskunft.

Eine kleine Besonderheit ist allerdings vergessen worden: Falls Sie einmal <CTRL+S> drücken, während ein Programm läuft, hält dieses sofort an. Damit haben wir eine Pausentaste im Computer. Erst nach einem beliebigen anderen Tastendruck arbeitet der Computer weiter.

GRAPHIC CLR

Eine Version des GRAPHIC-Befehls wurde Ihnen bisher noch verschwiegen. Mit GRAPHIC CLR führen Sie den Computer wieder in den reinen Textmodus zurück. Sie werden sagen, daß dazu doch die Befehle GRAPHIC0 oder GRAPHIC5 ausreichen. Das stimmt aber nur in Grenzen. Falls einmal einer der Grafik-Modi 1 bis 4 eingeschaltet wurde, damit also der ganze Speicher in BANK 0 verändert ist, bleibt die Bit-Map und der Basic-Start bei $4000 erhalten. Das erkennen Sie beispielsweise daran, daß – falls Sie nicht mit einer der Löschfunktionen das Bitmuster gelöscht haben – durch beispielsweise GRAPHIC1 auf dem Bildschirm immer noch das Grafikbild auftaucht, das Sie vorher einmal erstellt haben.

Erst durch den Befehl GRAPHIC CLR krempeln Sie die BANK 0 unseres Speichers wieder völlig um. Der Basic-Start wandert wieder nach $1C00, die Bit-Map wird überschrieben.

COLOR

Wir kommen nun zur Farbgebung in den verschiedenen Modi. Im Befehl

```
COLOR A,B
```

bedeutet A das folgende:

A=0 Hintergrund im 40-Zeichen-Textmodus und in den Grafik-Modi.

A=1 Vordergrundfarbe in den Grafik-Modi. Im Multicolormodus wird damit die Farbe der Bitkombination 01 belegt.

A=2 Multicolorfarbe 1. Das betrifft die Farbe der Bit-Kombination 10.

A=3 Multicolorfarbe 2, die sich auf die Bitkombination 11 bezieht.

A=4 Farbe des Bildschirmrahmens.

A=5 Zeichenfarbe im Textmodus. Damit wird sowohl im 40- als auch im 80-Zeichen-Text die Zeichenfarbe bestimmt.

A=6 Hintergrund des 80-Zeichen-Bildschirms.

B ist der Farbcode. Je nach Fabrikat, eingestelltem Kontrast, Helligkeit und anderen Gerätespezifica variiert die dargestellte Farbe. Der Code und die angegebenen Farben dienen daher eher als ungefährer Richtwert. Bild 7 zeigt die Zuordnung der verschiedenen Werte.

Sehen wir uns nun die Farbgebung etwas genauer an:

Die Farbspeicher

Sowohl einzelne Speicherstellen als auch größere RAM-Bereiche spielen eine Rolle bei der Farbzuordnung. Einen RAM-Bereich haben wir schon beim Textmodus erwähnt: Den Bildschirmfarbenspeicher zwischen 55296 und 56295 ($D800 bis $DBE7). Anders als beim VDC-Attribut-RAM spielen hier nur die Bits 0 bis 3 eine Rolle. Die oberen 4 Bit sind unbenutzt und unterliegen einem ständigen Wandel. Versuchen Sie einmal mittels Monitor und dem Kommando »M FD800« mehrmals hintereinander in dieses Farb-RAM zu sehen. Im Einschaltzustand finden Sie allerlei Werte zwischen $0D und $FD, denen aber allen das niederwertige Nibble $D eigen ist.

Als weiterer RAM-Bereich liegt – bisher noch nicht erwähnt – in den Grafik-Modi 1 bis 4 ein Farbspeicher für die Bit-Map vor, der von 7168 bis 8167 ($1C00 bis $1FE7) reicht. Parallel zur Einrichtung der Bit-Map sorgen die GRAPHIC-Befehle auch für die Zuordnung dieses Farb-RAM. Wie Sie sehen, enthält der Farbspeicher lediglich 1000 Byte Speicherplatz. Die Farbzuordnung geschieht also weiterhin wie im Textmodus, nämlich Pixel für Pixel (wobei jedes Pixel in der Form des 8 x 8-Rasters gehandhabt wird). So können Sie zwar auch im Hochauflösungsmodus in verschiedenen Farben auf den Bildschirm zeichnen (einfach, indem COLOR1 vor dem Zeichenbefehl eine andere Farbe erhält), sobald aber die Zeichnung genügend dicht an eine andere heranreicht, wechselt an der Stelle das gesamte Pixel seine Farbe, also auch der darin verlaufende Teil der alten Zeichnung.

Der Aufbau der einzelnen Bytes dieses RAM-Bereichs hängt von der Art des eingeschalteten Grafikmodus ab. In der Hochauflösungsgrafik enthält das LSN (das sind die Bits 0 bis 3, also das untere Nibble) einen Code für die Hintergrund-

Farbe	Code	Farbe	Code
Schwarz	1	Hellbraun	9
Weiß	2	Braun	10
Rot	3	Rosa	11
Türkis	4	Dunkelgrau	12
Violett	5	Grau	13
Grün	6	Hellgrün	14
Blau	7	Hellblau	15
Gelb	8	Hellgrau	16

Bild 7. Farben und Code-Nummern

farbe, wohingegen das MSN (die Bits 4 bis 7, das obere Nibble) den Code für die Vordergrundfarbe darstellt. Der verwendete Code ergibt sich jeweils aus dem Farbcode (vorhin als B bezeichnet) minus 1.

Im Mehrfarbenmodus dagegen finden wir im LSN die Multicolorfarbe 1 (als Farbcode-1). Das MSN bleibt unverändert.

Übrigens ist dieser Inhalt des Farb-RAM erst dann vorhanden, wenn auf irgendeine Weise ein Löschvorgang stattgefunden hat, beispielsweise durch GRAPHIC1,1. Vorher – also dann, wenn dieser Speicher lediglich durch GRAPHIC1 eingerichtet, aber nicht gelöscht wurde – können Sie dort nur wirres Byte-Durcheinander erkennen, falls Sie sich mittels dem Monitorkommando M den Speicherbereich ansehen.

Zwei Speicherstellen in der PAGE 3 korrespondieren mit dem eben vorgestellten Farb-RAM:
$3E2 (dezimal994) FG-BG
$3E3 (dezimal995) FG-MC1

FG-BG hängt mit der hochauflösenden, FG-MC1 mit der Multicolorgrafik zusammen. Im Einschaltzustand finden wir in diesen beiden Speicherzellen:

```
$3E2: $DB = bin.1101 1011
                ↑    ↑
                13 / 11
           h'grün-1 /d'grau-1
           Vordergr./Hintergr.
$3E3: $D1 = bin.1101 0001
                ↑    ↑
                13 /  1
           h'grün-1 / weiß-1
           Vordergr./Multicolor1
```

Mit diesen Werten wird das Farb-RAM belegt, sobald ein Grafikmodus 1 bis 4 eingeschaltet wird und ein Löschvorgang stattfindet. Sobald aber durch einen COLOR-Befehl andere Farben gewählt werden, drückt sich diese Änderung in diesen beiden Speicherstellen aus, und der nächste Löschvorgang belegt das Farb-RAM mit dem Inhalt der entsprechenden Speicherstelle.

Da ergibt sich allerdings ein Problem: Nehmen wir an, wir hätten den Grafikmodus 1 eingeschaltet und wollen nun in den Multicolormodus (3) umschalten. Das ist zwar ohne weiteres möglich, aber die Farben werden erst dann umgeschaltet (also das Farb-RAM erst dann mit dem Code aus $3E3 belegt), wenn ein Löschvorgang stattgefunden hat. Dann ist aber auch die Bit-Map gelöscht! Diese Schwierigkeit kann auf einigen Umwegen gelöst werden. Wie, das werden wir nachher noch sehen.

Vier Zeropage-Speicherstellen spielen in der Farbgebung eine wichtige Rolle: $83 bis $86 (dezimal 131 bis 134):
$83 COLSEL aktuelle Farbe
$84 MULTICOLOR1 Multicolorfarbe 1
$85 MULTICOLOR2 Multicolorfarbe 2
$86 FOREGROUND Vordergrundfarbe

Im Einschaltzustand findet man in $83 den Wert 0, in $84 eine 1 (also Farbcode von weiß-1), in $85 steht 2 (das ist rot-1) und in $86 liegt D (13, was hellgrün-1 bedeutet).

COLSEL enthält nach jedem Grafikbefehl die darin angegebene Farbquellenziffer (die wir noch kennenlernen).

Aus dem C 64-Modus sind Ihnen zwei Speicherstellen des VIC-Chip sicher bekannt:
$D020 dezimal 53280 Rahmenfarbe
$D021 dezimal 53281 Hintergrundfarbe

Auch im C 128-Modus haben die Speicherstellen diese Bedeutung. Im Einschaltzustand finden wir die Farben im LSN:
$D020 enthält $FD = binär 1111 1101
hellgrün-1
$D021 enthält $FB = binär 1111 1011
dunkelgrau-1

Nun kennen wir die beteiligten Speicher. Welche Auswirkungen hat der COLOR-Befehl?

Fassen wir die bis jetzt gewonnenen Erkenntnisse zur Farbgebung der Grafikbildschirme zusammen:

COLOR 0,X belegt den Hintergrund mit der Farbe X. Die Bits 0 bis 3 von $3E2 enthalten den Wert X-1.

Die Bits 0 bis 3 von $D021 werden ebenfalls mit X-1 beschrieben.

COLOR 1,X bestimmt die grafische Vordergrundfarbe. In $86 erscheint X-1.

Die Bits 4 bis 7 der Speicherstellen $3E2 und $3E3 enthalten danach X-1.

COLOR 2,X legt die Multicolorfarbe 1 fest. $84 hat nun X-1 zum Inhalt.

In den Bits 0 bis 3 der Speicherstelle $3E3 befindet sich ebenfalls X-1.

COLOR3,X bestimmt die Multicolorfarbe 2. In $85 befindet sich danach der Wert X-1.

Schließlich wird bei eingeschaltetem Grafikmodus 1 bis 4 durch einen Löschvorgang der Speicherbereich $1C00 bis $1FE7 (dezimal 7168 bis 8167) beschrieben mit
- dem Inhalt von $3E2 bei den Modi 1,2
- oder dem Inhalt von $3E3 in den Multicolor-Modi 3 und 4.

Wir werden nun ausprobieren, welche Bitpaarkombination welcher Farbe im Multicolormodus entspricht. Dazu schreiben wir an den Anfang der Bit-Map drei Zahlen, die diese drei möglichen Kombinationen repräsentieren:

```
binär 1111 1111 = dezimal 255 (Kombination 11)
binär 1010 1010 = dezimal 170 (Kombination 10)
binär 0101 0101 = dezimal  85 (Kombination 01)
```

Die Kombination 00 entspricht der Hintergrundfarbe. Das Programm MULTICOLORTEST1 (Listing 1) fragt zunächst nach den von Ihnen gewünschten Farben. Die Reihenfolge der Angaben folgt dabei den verschiedenen A-Kennungen des COLOR-Befehls, die wir vorhin verwendet haben. In Zeile 20 werden die von Ihnen gewünschten Farben dann in die Register geschrieben, in Zeile 30 der Mehrfarbenmodus eingeschaltet und das Farb-RAM beschrieben. Die Zeilen 40 bis 60 tragen die oben ermittelten Bitkombinationen in die ersten Plätze der Bit-Map ein. Die folgenden CHAR-Befehle (die lernen wir noch kennen) schreiben nun jeweils in einer der drei Farben einen Orientierungstext auf den Bildschirm. Übrigens sieht der Text manchmal in diesem Modus etwas merkwürdig aus: Die Zeichenmuster sind nämlich nicht für die Bitpaardarstellung entworfen.

Aus dem Programm folgt eine recht einprägsame Regel: Der Dezimalwert einer Bitkombination ist die Kennung A des dazugehörigen COLOR-Befehls. Erklärung:

Die Bitkombination 00 wird mittels COLOR0,... farblich festgelegt, die Bitkombination 10 (das ist dezimal 2) folgt in

```
5 REM ***** MULTICOLORTEST 1 *****
10 INPUT "FARBEN F0,F1,F2,F3";F0,F1,F2,F3
20 COLOR 0,F0: COLOR 1,F1: COLOR 2,F2: COLOR
   3,F3
30 GRAPHIC 3,1: SCNCLR 3
40 BANK 0: POKE DEC("2000"),255: POKE DEC("2
   001"),255
50 POKE DEC("2002"),170: POKE DEC("2003"),17
   0
60 POKE DEC("2004"),85: POKE DEC("2005"),85
70 CHAR 1,10,10,"OBEN :11"
80 CHAR 2,10,13,"MITTE:10"
90 CHAR 3,10,16,"UNTEN:01"
```

Listing 1. »MULTICOLORTEST1« – welche Bit-Kombination gehört zu welcher Farbquelle?

Bit:	7	6	5	4	3	2	1	0
bestimmt:	Vordergrund				Hintergrund			
Bit-Map-Inh.:	1				0			
Farbbefehl:	Color 1,...				Color 0,...			

Bild 8. Ein Byte aus dem Farb-RAM ab $1C00 in den Grafik-Modi 1 und 2

der Zeichnung der Farbe, die durch COLOR2,... definiert wird, und so weiter.
Nun können wir zusammenfassen:
Im hochauflösenden Modus folgt die Farbgebung dem Schema in Bild 8.
Im Mehrfarbenmodus kommt die Hintergrundfarbe (bei 00) aus der Speicherstelle 53281, die Farben der Kombinationen 01 und 10 werden gemäß dem Schema in Bild 9 verteilt.
Woher kommt die Farbe der Bitkombination 11? Wir haben festgestellt, daß diese Farbe in der Speicherstelle $85 festgehalten wird. Im C64-Modus wird sie aus dem normalen Text-Farb-RAM (ab $D800) entnommen. Überprüft man aber im C128-Modus diesen Speicherbereich, dann ist darin nur die normale Textfarbe zu finden. Hier muß also ein anderes Prinzip der Farbgebung verfolgt werden. Versuche zeigen, daß auch die Multicolorfarbe 2 pixelweise vergeben wird, was auf ein weiteres Farb-RAM schließen lassen könnte – aber die Suche danach verläuft ergebnislos. Diese Frage kann wohl erst dann geklärt werden, wenn eines Tages ein ROM-Listing des Basic-Interpreters zur Verfügung steht. Solange birgt unser Computer noch einige Geheimnisse.

Bit:	7	6	5	4	3	2	1	0
bestimmt:	Vordergrund				Multicolorfarbe 1			
Bit-Kombination:	01				10			
Farbbefehl:	Color 1,...				Color 2,...			

Bild 9. Ein Byte aus dem Farb-RAM ab $1C00 in den Grafik-Modi 3 und 4

Farbwechsel

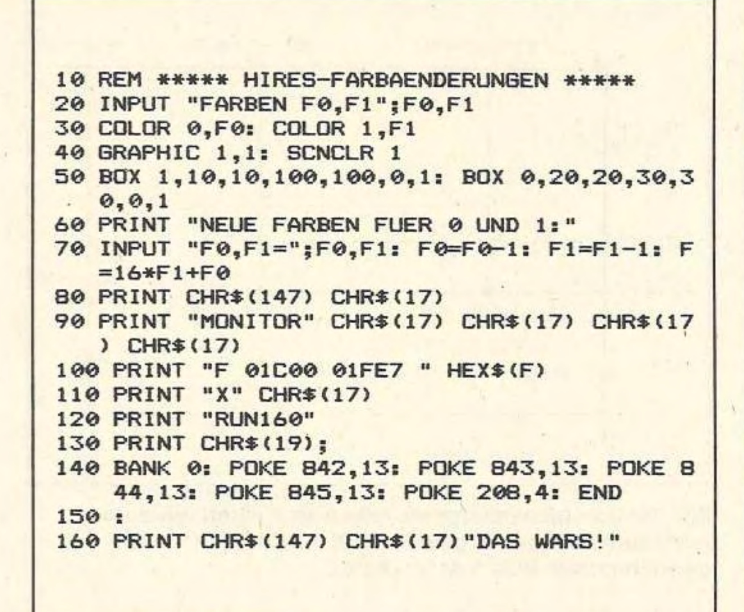
Zwei Aspekte sind es nun noch in bezug auf Farbgebung, die uns interessieren:
- Kann man mehr als vier Farben verwenden?
- Kann man in einem fertigen Bild die Farben ändern?

Der erste Aspekt ist sicher erfüllbar, denn ebenso, wie das Farb-RAM des Textbildschirms durch POKE-Befehle Zelle für Zelle ansprechbar ist, ist es auch das Grafik-Farb-RAM ab $1C00. Allerdings muß man sich gut überlegen, was man hier hineinschreibt: Sowohl die Bits 0 bis 3 als auch die Bits 4 bis 7 haben ja eine Bedeutung, wie wir gesehen haben.
Eine andere Möglichkeit ist es, mittels des COLOR-Befehls eine oder mehrere Farben (Vordergrund, Multicolor1 oder 2) zu ändern und dann mittels eines Grafik-Befehls eben diese Farbquelle aufzurufen (dazu kommen wir noch). Hier ist zu beachten, daß sich die grafischen Objekte, die damit gezeichnet werden, nicht zu nahe kommen, weil ja immer ein ganzes Pixel umgefärbt wird und daher auch schon vorhandene Bildteile unter Umständen die neue Farbe annehmen.
Wesentlich schwieriger ist es allerdings, ein Bild, dessen Farbgebung uns nicht gefällt, umzufärben. Nehmen wir an, ein Multicolor-Bildschirm trüge Farben, die wir verändern wollen. Was wäre zu tun?

```
5 REM ***** MULTICOLORTEST 2 *****
10 INPUT "FARBEN F0,F1,F2,F3";F0,F1,F2,F3
20 COLOR 0,F0: COLOR 1,F1: COLOR 2,F2: COLOR
   3,F3
30 GRAPHIC 3,1: SCNCLR 3
40 BANK 0: POKE DEC("2000"),255: POKE DEC("2
   001"),255
50 POKE DEC("2002"),170: POKE DEC("2003"),17
   0
60 POKE DEC("2004"),85: POKE DEC("2005"),85
70 CHAR 1,10,10,"OBEN :11"
80 CHAR 2,10,13,"MITTE:10"
90 CHAR 3,10,16,"UNTEN:01"
100 PRINT "NEUE FARBEN FUER 1 UND 2:"
110 INPUT "F1,F2=";F1,F2: F1=F1-1: F2=F2-1:
    F=16*F1+F2
120 PRINT CHR$(147) CHR$(17)
130 PRINT "MONITOR" CHR$(17) CHR$(17) CHR$(1
    7) CHR$(17)
140 PRINT "F 01C00 01FE7 " HEX$(F)
150 PRINT "X" CHR$(17)
160 PRINT "RUN200"
170 PRINT CHR$(19);
180 BANK 0: POKE 842,13: POKE 843,13: POKE 8
    44,13: POKE 845,13: POKE 208,4: END
190 :
200 PRINT CHR$(147) CHR$(17)"DAS WARS!"
```

Listing 2. »MULTICOLORTEST2« – ein fertiges Bild ändert seine Farben

Die Farbe aller Bitpaare 00 läßt sich einfach durch COLOR 0,... umschalten. Andere COLOR-Befehle für die drei restlichen Farben oder auch für die Vordergrund- und die Hintergrundfarbe im hochauflösenden Modus führen zu keinem Ergebnis: Erst ein Löschvorgang ergäbe die Neubelegung des Farb-RAM, wodurch aber auch die Bit-Map frei und unser Bild zerstört würde.
Da scheint nur eine Schleife Abhilfe zu bringen, die den Farbspeicher neu schreibt. Nehmen wir an, unsere neue Vordergrundfarbe hieße F1 und die neue Hintergrund- (in den Grafik-Modi 1 und 2) oder Multicolorfarbe 1 (im Grafikmodus 3 oder 4) wäre F2. Dann haben wir zunächst die Gestalt eines Bytes aus dem Farb-RAM zu bestimmen:
`F1=F1-1:F2=F2-1`
Das Byte wäre dann:
`16*F1+F2`
Die Schleife lautet nun:
`FOR I=7168 TO 8167:POKE I,16*F1+F2:NEXT I`
So etwas dauert natürlich eine Weile und wir können ganz gemächlich zusehen, wie von oben nach unten die Farbwechsel stattfinden.
Etwas schneller geht das mit einem Trick, der im Abschnitt zu den selbstmodifizierenden Programmen erklärt wird. Die beiden Programme MULTICOLORTEST2 (Listing 2) und HIRES-FARBAEND (Listing 3) zeigen Ihnen das Verfahren.
Beide Programme zeichnen zunächst etwas auf den Bildschirm und fragen dann nach den gewünschten neuen Farben. Falls Sie nur mit einem Bildschirm arbeiten, sollten Sie noch die Umschaltung auf den Textbildschirm (GRAPHIC0 vor der Frage) und hinterher wieder auf den Grafikbildschirm vornehmen. Nach dem Berechnen des Bytewertes schaltet das Programm den Monitor ein und füllt mittels der F-Funktion den Farbspeicher von $1C00 bis $1FE7. Danach verläßt es den Monitor wieder und fährt mit dem Basic-Programm fort (ab Zeile 160 beziehungsweise 200).
Auf diese Weise können alle beiden Farben im Hochauflösungsmodus verändert werden. Drei Farben von Multicolor-



```
10 REM ***** HIRES-FARBAENDERUNGEN *****
20 INPUT "FARBEN F0,F1";F0,F1
30 COLOR 0,F0: COLOR 1,F1
40 GRAPHIC 1,1: SCNCLR 1
50 BOX 1,10,10,100,100,0,1: BOX 0,20,20,30,3
   0,0,1
60 PRINT "NEUE FARBEN FUER 0 UND 1:"
70 INPUT "F0,F1=";F0,F1: F0=F0-1: F1=F1-1: F
   =16*F1+F0
80 PRINT CHR$(147) CHR$(17)
90 PRINT "MONITOR" CHR$(17) CHR$(17) CHR$(17
   ) CHR$(17)
100 PRINT "F 01C00 01FE7 " HEX$(F)
110 PRINT "X" CHR$(17)
120 PRINT "RUN160"
130 PRINT CHR$(19);
140 BANK 0: POKE 842,13: POKE 843,13: POKE 8
    44,13: POKE 845,13: POKE 208,4: END
150 :
160 PRINT CHR$(147) CHR$(17)"DAS WARS!"
```

Listing 3. »HIRES-FARBAEND« – ein Bild in den Grafik-Modi 1 und 2 ändert seine Farben

bildern sind ebenfalls auf diese Art und mit COLOR 0,... zu erneuern. Interessant ist es übrigens, daß trotz des veränderten Farb-RAM ein Zeichenbefehl (beispielsweise DRAW1,...) immer noch mit der alten Zeichenfarbe arbeitet. Die Erklärung liegt vermutlich darin, daß dazu die Speicherstellen ab $83 Verwendung finden. Wenn man auch die darin befindlichen Angaben der neuen Farbgebung anpaßt, geschieht das nicht mehr.

Wie schon vorhin angedeutet, ist die Änderung der Multicolorfarbe 2, von der noch nicht die Herkunft geklärt werden konnte, ein Problem. Bisher existiert nur eine etwas umständliche Lösung: Speichern Sie Ihre Grafik inklusive des Farb-RAM ab. Das kann mittels des BSAVE-Befehls geschehen:
```
BSAVE"Bild",ON B0,P7168 TO P16191
```
Laden Sie nun das Bild wieder ein mittels:
```
COLOR3,X:GRAPHIC3,1:BLOAD"Bild"
```
In X steht dabei die neue gewünschte Multicolorfarbe 2, die nun gleich beim Einladen berücksichtigt wird.

DRAW

Nach all diesen etwas ermüdende aber notwendigen Vorbereitungen kommen wir nun zu den Zeichenbefehlen, deren einfachster der DRAW-Befehl ist.
```
DRAW A,X1,Y1 TO X2.Y2
```
zeichnet eine Linie vom Punkt X1,Y1 zum Punkt X2,Y2 (siehe Bild 10).

A ist eine Kennzahl für die Quelle, aus der die Farbe der Zeichnung herrührt. A hat dabei folgende Bedeutung:
A = 0 Hintergrundfarbe
A = 1 Vordergrundfarbe
A = 2 Multicolorfarbe 1
A = 3 Multicolorfarbe 2

Dieser Befehl ist sehr vielseitig verwendbar. Es ist beispielsweise möglich, einen Linienzug über weitaus mehr als nur zwei Punkte zu führen. Der Befehl
```
DRAW1,X1,Y1 TO X2,Y2 TO X3,Y3 TO X4,Y4 TO X5,Y5
TO X1,Y1
```
kann so ein grafisches Gebilde wie in Bild 11 erzeugen. Lautet der Befehl
```
DRAW1,X1,Y1
```

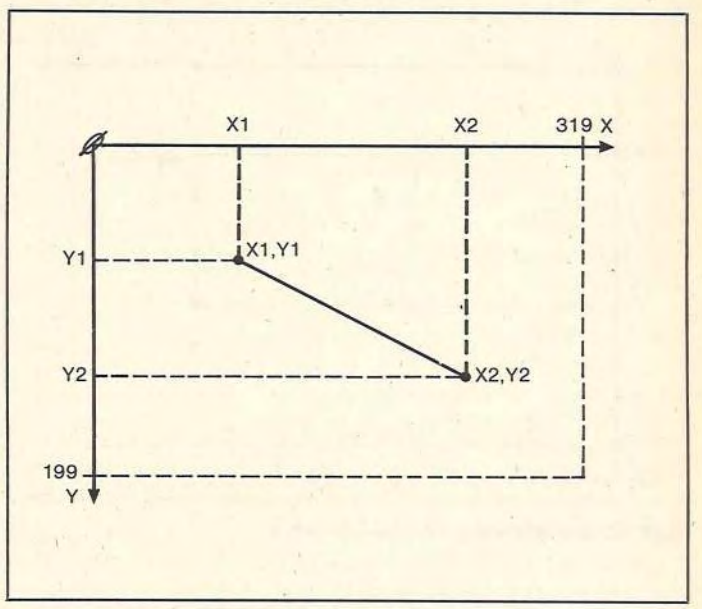


Bild 10. Die Wirkung des DRAW-Befehls

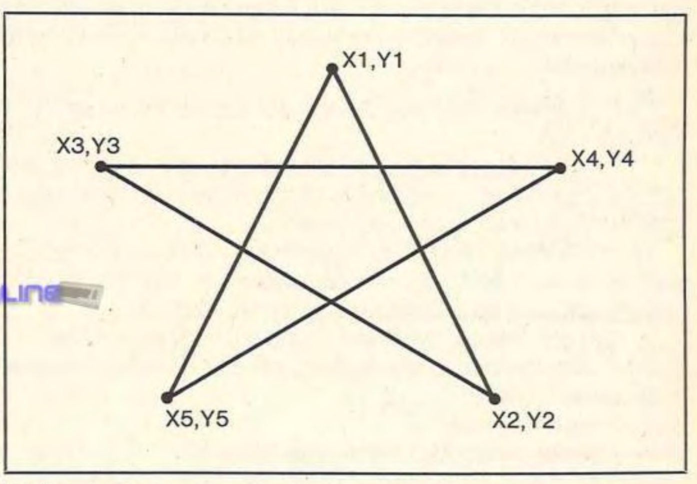


Bild 11. Beispiel einer durch einen DRAW-Befehl erzeugten Figur

dann wird lediglich der Punkt X1,Y1 gezeichnet.
Schreibt man dagegen den Befehl als
```
DRAW 1 TO X1,Y1
```
dann dient als Ausgangspunkt die aktuelle Grafikcursorposition. Den Grafikcursor werden wir beim LOCATE-Befehl noch kennenlernen. Von dieser Position aus wird dann eine Linie nach X1,Y1 gezogen.

Nach der Ausführung der verschiedenen DRAW-Befehle befindet sich der Grafikcursor an dem Punkt, der durch die letzte Koordinatenangabe charakterisiert wird.

Läßt man übrigens die Farbkennung weg, schreibt also beispielsweise
```
DRAW ,X1,Y1 TO X2,Y2
```
dann wird die Linie in der Farbe gezeichnet, die im vorangegangenen Zeichenbefehl verwendet wurde. Vielleicht erinnern Sie sich noch an die Speicherstelle $83 COLSEL? Dort befindet sich immer noch vom vergangenen Befehl her die Farbkennung.

BOX

Mit diesem Zeichenbefehl können Rechtecke erzeugt werden. Auch hier sind wieder eine ganze Menge Variationen möglich. Die Syntax des BOX-Befehls ist:
```
BOX A,X1,Y1,X2,Y2,W,F
```

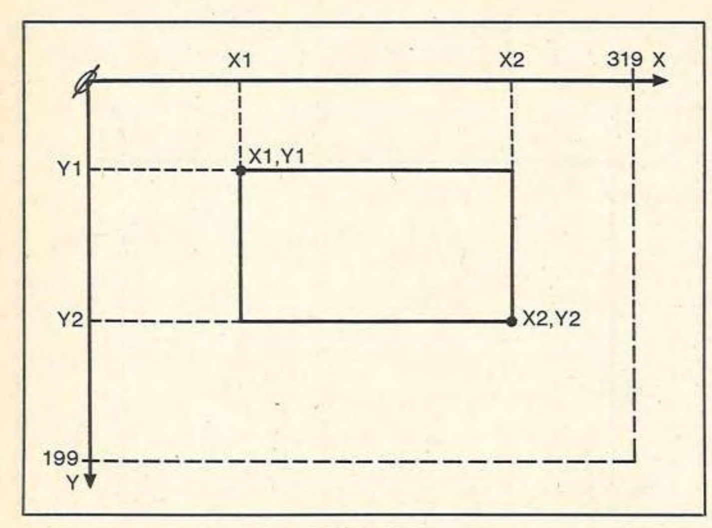


Bild 12. Die Wirkung des BOX-Befehls

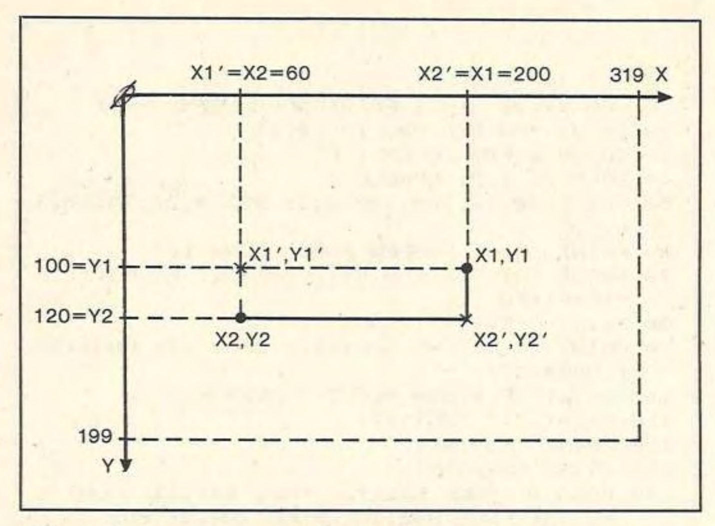


Bild 14. Durcheinandergewürfelte Koordinaten führen dennoch zum richtigen Ergebnis: BOX 1,X2,Y2,X1,Y1 wird gezeichnet wie BOX 1,X1',Y1',X2',Y2'

A ist wieder - wie schon beim DRAW-Befehl - die Farbquelle. Ebenso wie dort kann man A auch weglassen und das Rechteck erhält dieselbe Farbe wie das zuvor gezeichnete Grafikobjekt.

Bild 12 erläutert die Bedeutung der Eckkoordinaten X1,Y1 und X2,Y2.

X1,Y1 sind also die Koordinaten der linken oberen Ecke, X2,Y2 diejenigen der rechten unteren. Wir werden darauf gleich noch einmal zurückkommen.

W ist ein Winkel (in Grad angegeben!), um den das Rechteck zu drehen ist. Die Drehung erfolgt um den Mittelpunkt des Rechtecks im Uhrzeigersinn (siehe Bild 13).

W darf alle Werte zwischen 0 und 65535 annehmen.

F ist eine Kennung, die festlegt, ob das Rechteck ausgefüllt werden soll:

F = 0 Nicht ausfüllen

F = 1 In der gewählten Farbe ausfüllen.

Interessant ist, was geschieht, wenn man Koordinaten eingibt, die unlogisch erscheinen. Beispielsweise in

```
BOX 1,200,100,60,60
```

wurden die Eckpunkte vertauscht. Der Computer wählt sich selbst die richtigen Ecken aus und zeichnet die Gestalt, die wir bei der richtigen Reihenfolge erhalten hätten, also

```
BOX 1,60,60,200,100
```

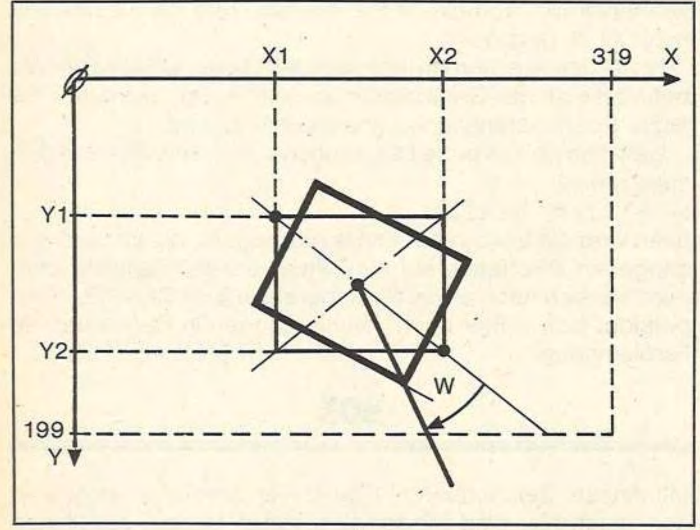


Bild 13. Drehung des Rechtecks durch den Parameter W im BOX-Befehl

Würfeln wir die Koordinaten vollends durcheinander (siehe Bild 14).

```
BOX 1,200,100,60,120
```

gibt die linke untere und die rechte obere Ecke an. Gezeichnet wird vom Computer dennoch das richtige Rechteck (mit den Ecken X1',Y1' und X2',Y2'), als hätten wir ihm befohlen

```
BOX 1,60,100,200,120
```

Es scheint, als gäbe es keinen Unterschied bei den verwürfelten Koordinatenangaben. Da existiert aber doch einer! Wir haben bisher den Grafikcursor nicht berücksichtigt. Der steht nämlich immer nach der Zeichnung am Punkt X2,Y2. Das bedeutet, daß er im Fall

```
BOX 1,200,100,60,120
```

auf dem Punkt 60,120 steht, wohingegen er im anderen Fall bei 200,120 zu finden wäre.

Bei einer Drehung des Rechtecks durch einen Wert W wird der Grafikcursor nicht mitgedreht. Er bleibt stur bei X2,Y2 stehen.

Noch nicht ausdrücklich erwähnt wurde die Möglichkeit, für den Fall, daß sie nicht benötigt werden, die Größen W und F einfach wegzulassen. Sie werden dann beide auf 0 gesetzt. Man kann aber auch X2 und Y2 weglassen. Das zu zeichnende Rechteck setzt dafür einfach wieder die Koordinaten des Grafikcursors ein.

Ein Beispiel:

```
BOX 3,100,50,200,80,30,1
```

zeichnet in der Multicolorfarbe 2 (dazu muß natürlich der Grafikmodus 3 eingeschaltet sein) ein Rechteck, das um 30 Grad um seinen Mittelpunkt im Uhrzeigersinn gedreht wurde und ausgefüllt ist.

CIRCLE

Ein geradezu phantastisch vielseitiger Befehl, nicht nur zum Zeichnen von Kreisen, wie es der Name (circle ist das englische Wort für Kreis) suggeriert. Was man damit alles anstellen kann, werden Sie gleich noch sehen. Wieder kann man über eine lange Parameterliste alle Optionen ausschöpfen. Aber keine Angst, es müssen bei weitem nicht immer alle angegeben werden. Komplett sieht der Befehl so aus:

```
CIRCLE A, XM,YM, RX,RY, W1,W2, W, S
```

Der erste Parameter A dient wie schon bei den anderen Zeichenbefehlen zur Festlegung der Farbquelle.

Die Koordinaten XM,YM legen den Mittelpunkt des Objekts fest, das wir zu zeichnen gedenken. Soll es ein Kreis werden, ist es also der Kreismittelpunkt.

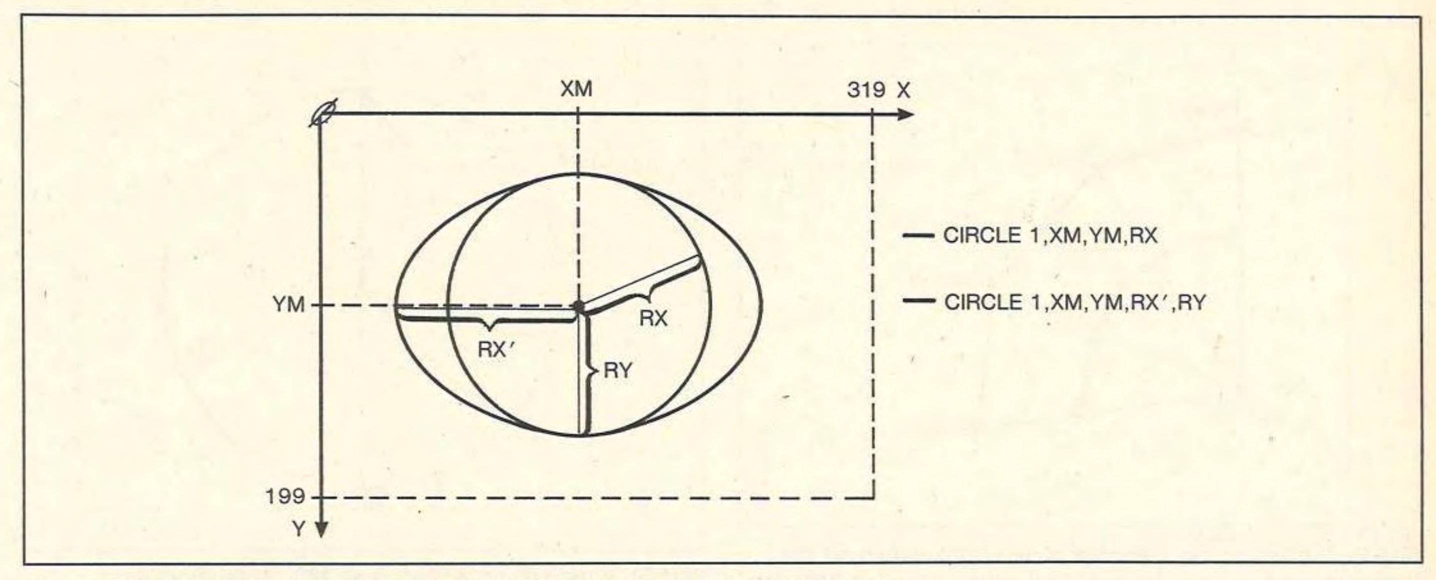


Bild 15. Wirkung der Befehle CIRCLE 1,XM,YM,RX und CIRCLE 1,XM,YM,RX',RY

RX und RY sind die Halbmesser in horizontaler und vertikaler Richtung. Gedanklich geht man dabei von der Ellipse aus, die ja zum Kreis wird, wenn beide Radien gleich sind. In diesem Fall ist RX = RY und man kann sogar RY weglassen. Bild 15 zeigt, wie die bisher verwendeten Bezeichnungen zu verstehen sind.

Zwei CIRCLE-Befehle erzeugen hier einen Kreis (CIRCLE 1,XM,YM,RX) und eine Ellipse (CIRCLE 1,XM,YM,RX',RY).

Die Angaben W1 und W2 erlauben die Festlegung eines Bogens aus dem Gesamtobjekt, der zu zeichnen ist. Ausgangspunkt für die Messung der Winkel W1 und W2 ist der obere Scheitelpunkt der Ellipse, die man sich durch die Figur gelegt denken kann. Die Formulierung hört sich wesentlich einfacher an, wenn man von der Zeichnung eines Kreises ausgeht, der im normalen Koordinatensystem liegt. Dann zählt der Winkel vom obersten Kreispunkt an im Uhrzeigersinn, was Bild 16 erläutern soll.

W1 und W2 dürfen ohne Fehlermeldung alle Werte zwischen 0 und 65535 annehmen, wobei allerdings Angaben, die größer als 33024 sind, allerlei Unsinn auf dem Bildschirm erzeugen.

Was geschieht, wenn W1 größer als W2 gewählt wird? Kein Problem: Das Ergebnis finden Sie in einem Beispiel in Bild 17.

Es wird der Bogen von W1 über 0 hinweg bis W2 gezeichnet.

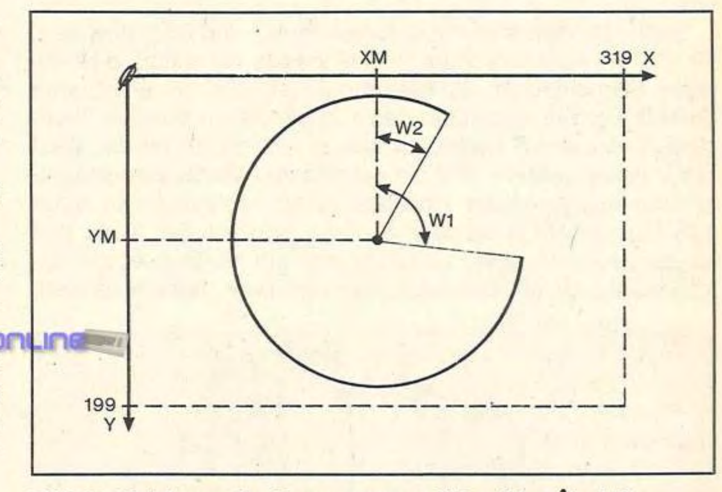


Bild 17. Zeichnen des Bogens, wenn W1 größer als W2

Ein weiterer Parameter ist W, der Drehwinkel. Ebenso wie beim BOX-Befehl kann auch das durch CIRCLE erzeugte Objekt um den Mittelpunkt im Uhrzeigersinn gedreht werden. Bild 18 zeigt die Verhältnisse dabei.

Ohne zu murren, nimmt der Computer Werte zwischen 0 und 65535 für W entgegen.

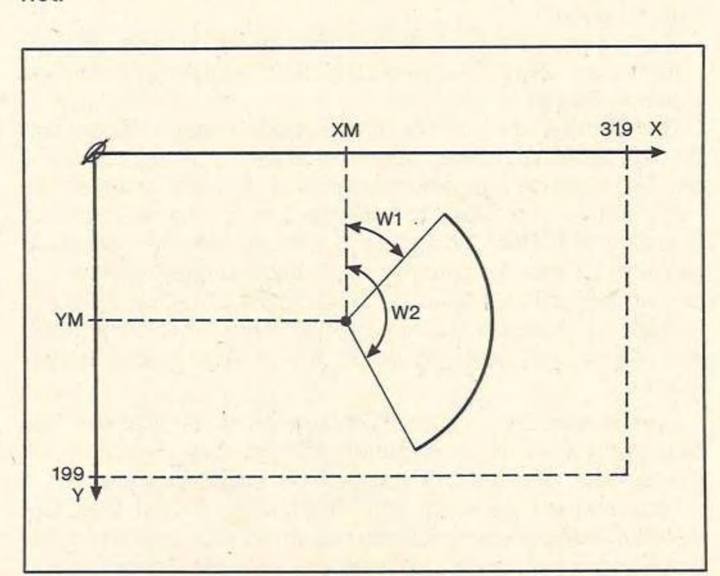


Bild 16. Zeichnen eines Kreis-(oder Ellipsen-)Bogens durch CIRCLE 1,XM,YM,RX,RY,W1,W2

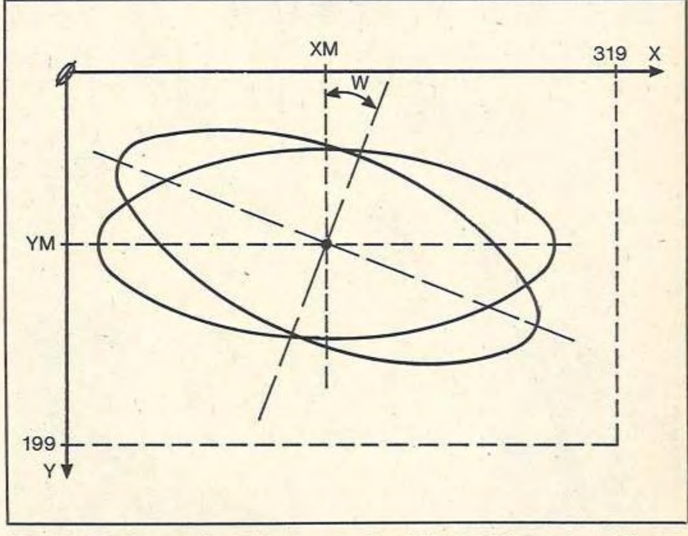


Bild 18. Drehung der Ellipse um den Winkel W um den Mittelpunkt XM,YM durch: CIRCLE 1,XM,YM,RX,RY,,,W

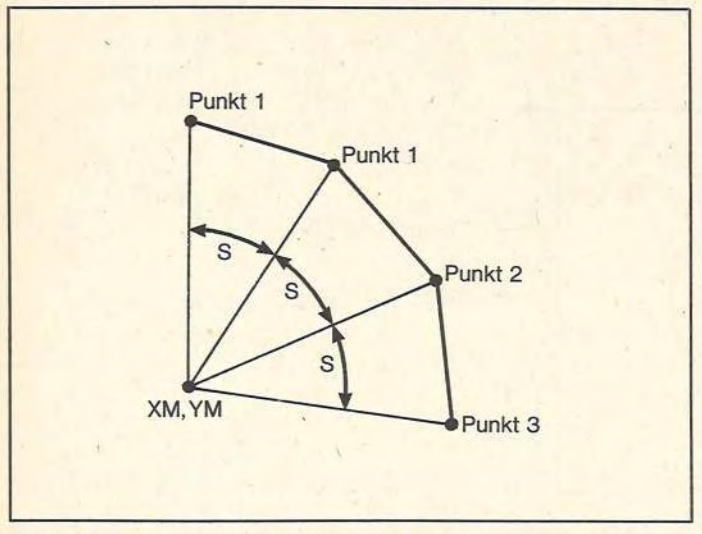


Bild 19. Der Parameter S bestimmt den Winkelabstand der berechneten Kreispunkte, zwischen denen Geraden gezogen werden

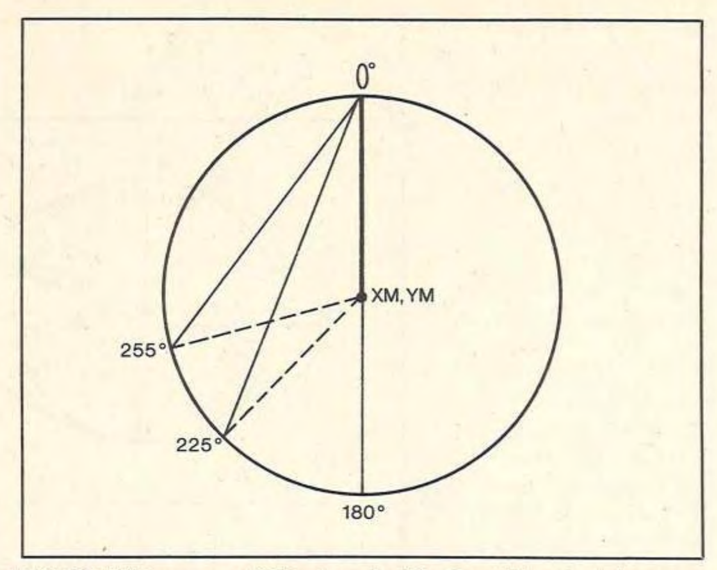
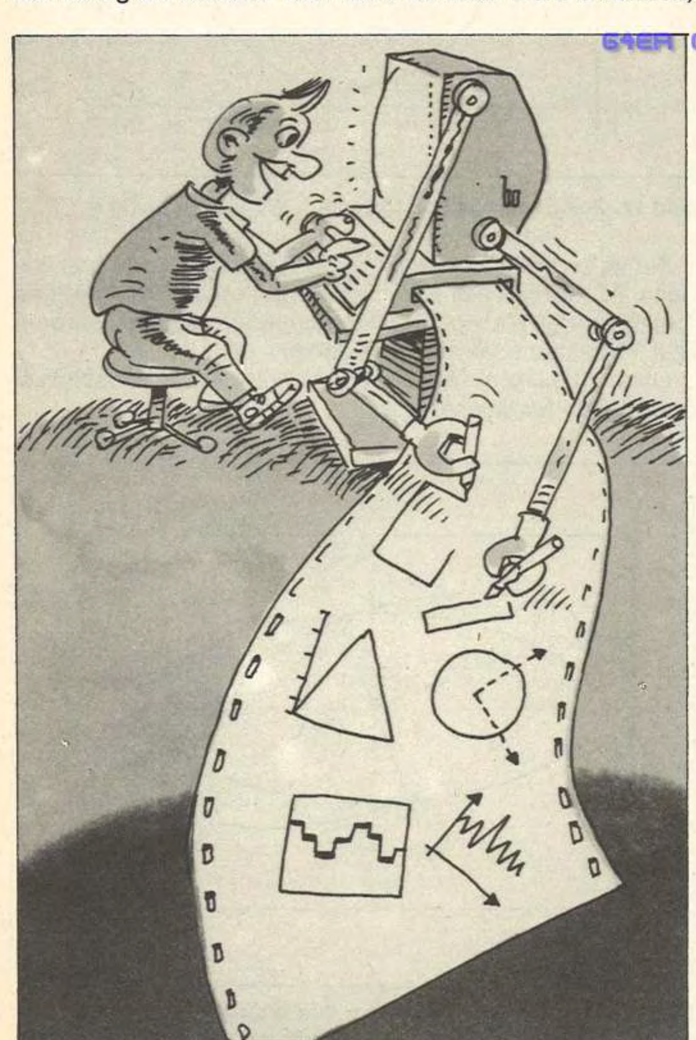


Bild 20. Wirkung von S-Werten ab 180. Gezeichnet wird jeweils die dickere Linie.

Der letzte Parameter S kann zwischen 1 und 255 groß sein. Er gibt an, nach welchem Winkel jeweils der nächste Kreis- oder Ellipsenpunkt zu berechnen ist. Die so ermittelten Punkte werden durch Geraden miteinander verbunden. Niedrige Werte von S bedeuten kleine Winkel. So ist der Wert S=2 voreingestellt und die miteinander verbundenen Eckpunkte ergeben den Eindruck eines Kreises (oder einer Ellipse). Erhöht man aber S, dann beginnt der Kreis (ich erspare mir im weiteren den Hinweis auf die Ellipse) allmählich kantig zu werden. Man kann es zwar nicht erkennen, trotzdem ist der schnell gezeichnete Kreis (oder...) in Wahrheit ein 36-Eck, wenn S=10 gewählt wurde. Weitere Polygone ergeben sich zum Beispiel:

```
S=1  360-Eck
S=2  180-Eck
S=10 36-Eck
S=20 18-Eck
S=30 12-Eck
S=60  6-Eck
```

Das Bild 19 soll dazu dienen, diese Angelegenheit zu illustrieren.



Weil der gesamte Kreis einen Winkel von 360 Grad überstreicht, läßt sich eine Formel angeben, mit der gezielt n-Ecke erzeugt werden können:

S = 360/n

Wenn Sie also ein Dreieck (n=3) erzeugen möchten, müssen Sie S=120 eingeben. Die Formel zeigt uns noch zwei weitere Eigenschaften von S:

- Zum einen kann man sie umdrehen und berechnen, wieviele Ecken ein Polygon bei einem bestimmten S-Wert hat:
  n = 360/S

Dabei zeigt es sich, daß für »n« auch allerhand gebrochene Zahlen zustandekommen. Weil es aber beispielsweise kein 7,2-Eck gibt (bei S=50), muß es sich hier um ein krummes 7-Eck handeln.

- Zum zweiten sind Eingaben größer als 180 sinnlos. S=180 führt zum »Zweieck«, also zum Durchmesser des Kreises (siehe Bild 20).

Der Startpunkt ist immer oben (also bei einem Winkel von 0 Grad) zu finden, der Endpunkt liegt auf der Kreisperipherie um 180 Grad demgegenüber verdreht. Ein Weiterzählen um noch einmal 180 Grad führt wieder zum Startpunkt. Wächst S auf noch höhere Werte, dann wird nur noch eine Sehne gezeichnet vom Startpunkt zum Punkt 1, der beispielsweise bei S=225 um 225 Grad, bei S=255 um 255 Grad verdreht auftritt. Ein weiterer Punkt wird hier nicht mehr berechnet, weil damit der Ausgangspunkt bei 0 Grad überschritten würde.

Den Grafikcursor hätten wir fast noch vergessen: Der befindet sich (im Gegensatz zum BOX-Befehl) immer dort, wo der letzte Punkt des CIRCLE-Befehls gesetzt wurde.

Übrigens gilt es noch eine Besonderheit des CIRCLE-Befehls im Multicolormodus zu beachten. Gibt man hier nämlich einen Befehl zum Zeichnen eines Kreises in der Form

```
CIRCLE 1,XM,YM,RX
```

ein, wobei ja automatisch RY = RX gesetzt wird, dann ent-

steht zwar auf dem Bildschirm ein Kreis. Der ist aber falsch, was Sie unschwer erkennen können, indem Sie oben und unten Tangenten anlegen. Das Ganze sähe beispielsweise so aus:

```
CIRCLE 1,80,100,40:DRAW 1,0,40 TO 159,40
DRAW 1,0,120 TO 159,120
```

Als gebildete Programmierer sollten wir erwarten, daß diese Geraden als Tangenten den Kreis am oberen und am unteren Rand berühren. Aber weit gefehlt, sie schneiden ihn! Probieren wir dagegen einmal, den Kreis mittels

```
CIRCLE 1,80,100,40,40
```

zu zeichnen, dann sollte ja dasselbe herauskommen. Aber das Ergebnis dieser Zeile ist eine Ellipse! Woran liegt das? Die Ursache dafür liegt in der ungleichen Skalierung der X- und der Y-Achse im Multicolormodus. Wenn wir dem Computer den Auftrag erteilen, einen Kreis zu zeichnen, dann zeichnet er ihn – ohne Rücksicht auf Verluste oder Skalierungen oder ähnliche Feinheiten.

PAINT

Damit kann eine von Kurvenzügen umschlossene Fläche ausgefüllt werden. Sie muß aber auch wirklich völlig eingerahmt sein, denn das kleinste Loch in der Hülle führt dazu, daß auch die benachbarte Fläche bemalt wird. Im Befehl

```
PAINT A, X1,Y1, M
```

haben die einzelnen Größen folgende Bedeutung:

A ist – wie schon gehabt – wieder die Farbquelle, mit der das Ausmalen geschehen soll.

X1 und Y1 sind die Koordinaten eines Punktes, der in der auszufüllenden Fläche enthalten ist. Gleichzeitig ist das der Startpunkt der Aktion und hinterher liegt genau hier der Grafikcursor.

Ein etwas schwerer verständlicher Parameter ist der Modus M. Ist nämlich M gleich 0, dann wird solange ausgefüllt, bis eine Umrahmung in der Farbe angetroffen wird, die im Parameter A gewählt wurde. Ist dagegen M gleich 1, dann zählt auch jede andere Farbe der Umrandung – sofern sie nur nicht die des Hintergrundes ist – als Grenze des Ausmalens. Befinden wir uns beispielsweise im Grafikmodus 3, dann passiert bei der Befehlskombination:

```
BOX2,10,10,100,100:CIRCLE3,80,100,40:PAINT 1,85,85
```

nicht das Ausmalen der Schnittmenge von Kreis und Rechteck, sondern ungerührt um unsere Bemühungen übermalt der Computer alles, was auf dem Bildschirm zu sehen war. Der Grund ist, daß keine Umrahmung in der Farbe mit der Kennung A = 1 vorhanden war (BOX lief mit A=2 und CIRCLE mit A=3) und wir den Modus 0 gewählt haben. Modus 1 hätte unseren Wunsch erfüllt.

PAINT arbeitet im Modus M=1 nicht, wenn am Startpunkt schon eine andere Farbe als die Hintergrundfarbe gefunden wird. Im Modus M=0 dagegen wird immer dann ausgefüllt, wenn dort eine Farbe vorhanden, die von der ausgewählten verschieden ist.

CHAR

Eigentlich kein Grafikbefehl im engeren Sinn ist dieser Befehl zum Drucken eines Textes. Seine interessanteste Anwendung ist aber die Beschriftung von Grafiken, was auf andere Weise (außer in den Splitscreen-Modi) nicht möglich ist.

```
CHAR A, XT,YT, A$, R
```

bildet einen Text A$ in der Farbe der Quelle A ab. Das erste Zeichen befindet sich bei den Koordinaten XT,YT. Das Koordinatensystem, das hier eine Rolle spielt, ist das im Textmodus gültige (25 Zeilen zu je 40 Spalten oder aber 80 Spalten). A$ kann jede beliebige Kombination von gültigen Zeichen enthalten. In den Grafik-Modi werden allerdings die Steuerzeichen als grafische Zeichen mitgedruckt.

R ist eine Kennzahl, die es erlaubt, den Text normal oder revers darzustellen:

R = 0 Normaldarstellung
R = 1 Reversdarstellung

Sollte der Text nicht in einer Zeile Platz finden, dann wird er in der nächsten fortgesetzt, also ein Verhalten, wie wir es auch vom PRINT-Befehl her kennen.

Im Multicolormodus sehen die Zeichen häufig etwas merkwürdig aus. Die Zeichenmuster sind ja nicht zur Darstellung im Bitpaarmodus konstruiert. Soll ein Text in der Multicolorfarbe 2 gedruckt werden, dann ist A auf 0 und R auf 1 zu setzen. Soll dagegen die Multicolorfarbe 1 verwendet werden, hält man die Farbquelle A auf 1 und für R wählt man den Wert 0. So steht es jedenfalls im Handbuch (auf den Seiten 4-28 und 4-29). Zumindest mein C 128 folgt diesem Rezept nicht, sondern es genügt, einfach A entsprechend der gewünschten Farbquelle (also 0,1,2,3) zu wählen.

Der Grafikcursor bleibt übrigens vom CHAR-Befehl unbeeindruckt.

LOCATE

Wir haben bei den bisher untersuchten Befehlen immer wieder einen Grafikcursor und dessen Verhalten bei den verschiedenen Operationen erwähnt. Sie haben diesen Cursor aber noch nie zu Gesicht bekommen – werden Sie auch nie, denn er führt sein verborgenes Leben nur in einigen Speicherstellen, nie aber auf dem Bildschirm. Trotzdem ist er recht nützlich. Er dient gewissermaßen als Merkstelle für bestimmte Punktkoordinaten:

- Als Startpunkt bei DRAW
  Beispielsweise für DRAW 1 TO X2,Y2
- Als Eckpunkt bei BOX
- Als Mittelpunkt bei CIRCLE: Laut Handbuch soll man die Koordinaten XM,YM weglassen können und dafür werde automatisch der Ort des Grafikcursors eingesetzt. Leider war das bei meinem C 128 nicht nachvollziehbar. Es ergab sich nur allerlei Unsinn auf dem Bildschirm.
- Als Startpunkt für PAINT: Auch hier kann man ohne Angabe der Startpunktkoordinaten Flächen ausmalen, denn es wird automatisch der Ort des Grafikcursors verwendet.
  LOCATE X,Y führt diesen Grafikcursor an den durch die Koordinaten X,Y definierten Punkt.

RDOT

Wollen Sie erfahren, wo sich der Grafikcursor im Augenblick befindet, dann dient dieser Befehl Ihrem Wunsch. Das Argument von RDOT(n) hat folgende Bedeutungen:

n = 0 liefert die aktuelle X-Position.
n = 1 ergibt die aktuelle Y-Position des Grafikcursors.
n = 2 macht eine Aussage über die aktuelle Farbquelle.

Die Koordinatenwerte, die RDOT-Anfragen ergeben, beziehen sich immer auf die grundlegenden Koordinatensysteme. Im hochauflösenden Grafikmodus stammen die Werte daher immer aus einem System, dessen X-Werte zwischen 0 und 319 liegen und dessen Y-Werte von 0 bis 199 reichen. Im

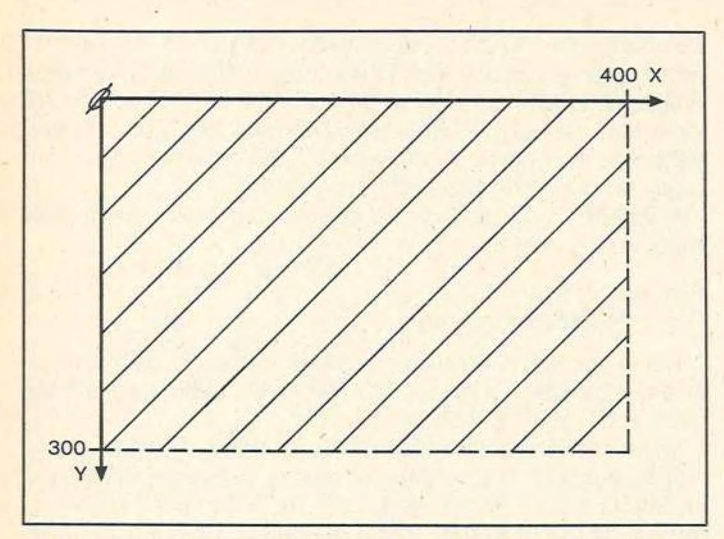


Bild 21. Wirkung von SCALE 1,400,300 in den Grafik-Modi 1 und 2

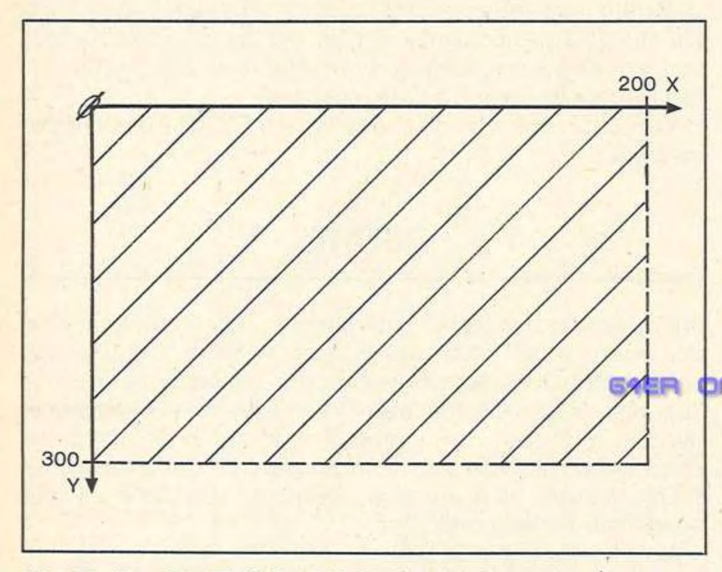


Bild 22. So wirkt SCALE 1,400,300 in den Grafik-Modi 3 und 4

Multicolormodus geht die X-Achse nur bis 159. Das im Auge zu behalten ist besonders für den nun folgenden Befehl wichtig:

SCALE

Mittels

```
SCALE n, Xmax, Ymax
```

können wir neue Koordinatensysteme auf unserem Bildschirm definieren. Die Kennzahl n kann zwei Optionen anwählen:

n = 0 Das reguläre System ist eingeschaltet:
Grafik-Modi 1 und 2:
X von 0 bis 319, Y von 0 bis 199
n = 1 Ein verändertes System ist nun Bezugssystem geworden.

Wenn durch n=1 die neue Skalierung eingeschaltet wurde, gibt es zwei Möglichkeiten. Schreibt man lediglich SCALE 1, dann hat man ein Koordinatensystem gewählt, dessen X- und Y-Koordinaten jeweils von 0 bis 1023 reichen. Der zweite Weg ist es, selbst die Reichweiten der Skalen festzulegen. Das geschieht durch die Angaben von Xmax und Ymax, den Höchstwerten auf der jeweiligen Achse. So wählt im Grafikmodus 1 der Befehl

```
SCALE 1, 400, 300
```

ein System, dessen X-Achse von 0 bis 400 und Y-Achse von 0 bis 300 reicht (siehe Bild 21).

Die Werte Xmax müssen größer als 320 gewählt werden, die für Ymax größer als 200. Es ist so also nicht möglich, verkleinerte Systeme zu erzeugen. Im Gegensatz zu den Handbuchangaben, die nur Werte bis 1023 erlauben, ist es tatsächlich möglich, sowohl Xmax als auch Ymax bis 32767 anzugeben.

Im Multicolormodus ist zu beachten, daß weiterhin auf der X-Achse immer Bit-Paare dargestellt werden. Im Grafikmodus 3 oder 4 erzeugt daher der Befehl

```
SCALE 1, 400, 300
```

ein System, dessen X-Koordinaten von 0 bis 200 und dessen Y-Koordinaten von 0 bis 300 reichen (siehe Bild 22).

Natürlich werden weiterhin die Abbildungen auf dem Bildschirm mit einer Auflösung von insgesamt 64000 Punkten (oder 32000 im Multicolormodus) dargestellt. Die Bit-Map bleibt unverändert. Lediglich der Berechnungsweg zur Darstellung der Punkte verändert sich. Etwas unglücklich werden viele Anwender des C 128 darüber sein, daß der SCALE-Befehl keine weitergehenden Umstellungen des Koordinatensystems ermöglicht. Wünschenswert wäre sicherlich die Möglichkeit gewesen, den Ursprung zu verschieben (wie es der TRS-Befehl der Grafikerweiterung HIRES-3 für den C64-Modus erlaubt). Durch diese mindere Ausstattung kommt dem SCALE-Befehl nicht die große Bedeutung zu, die er haben könnte, denn jeder Anwender muß die langwierigen Transformationen weiterhin programmieren.

SCALE hat auch Einfluß auf das Ergebnis eines CIRCLE-Befehls. Ebenso, wie wir es vorhin beim Multicolormodus beobachten konnten, wo ohne Rücksicht auf den anderen Maßstab der Y-Achse ein Kreis gezeichnet wurde, findet das hier in allen Grafik-Modi statt. Wenn also der richtige Kreis – das heißt derjenige, der bei allen Maßstäben in Y-Richtung den gleichen Durchmesser aufweist wie in X-Richtung – gezeichnet werden soll, muß das nun immer durch den CIRCLE-Befehl mit zwei Radienangaben geschehen. Das sollen Ihnen die Bilder 23 und 24 erläutern.

Die Verhältnisse in diesen Abbildungen beziehen sich auf die Grafik-Modi 1 und 2. Das Achsenverhältnis im Multicolormodus ist durch den Faktor 2 in X-Richtung verändert. Man sollte also festhalten, daß immer dann unverzerrte Bilder zustandekommen, wenn die Achsenverhältnisse wie folgt sind:

Grafik-Modi 1 und 2:X:Y = 1.6
Grafik-Modi 3 und 4:X:Y = 3.2

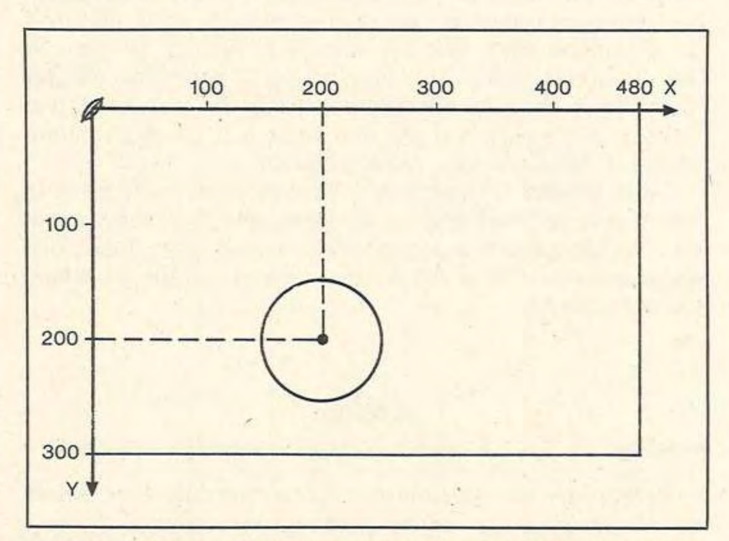


Bild 23. SCALE 1,480,300 erzeugt in den Grafik-Modi 1 und 2 dasselbe Achsenverhältnis wie das normale System: 320/200 = 480/300 = 1.6. Der Kreis CIRCLE 1,200,200,50 ist identisch mit dem Kreis CIRCLE 1,200,200,50,50.

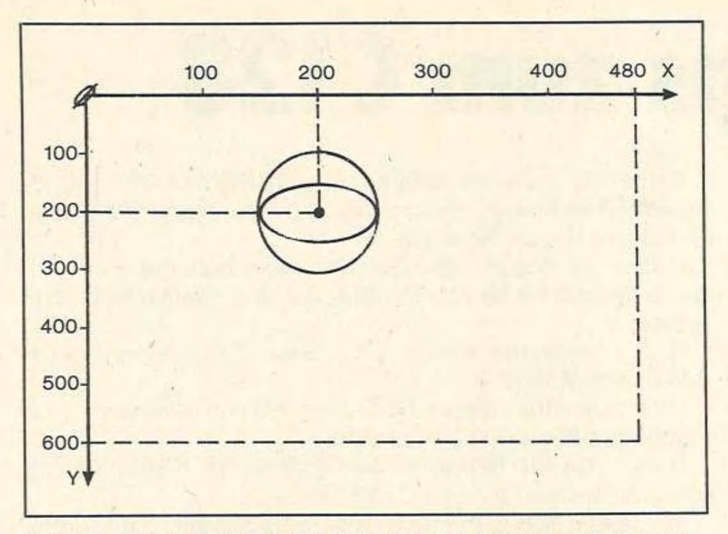


Bild 24. Durch SCALE 1,480,600 ist ein Achsenverhältnis von 480/600 = 0.8 geschaffen worden. CIRCLE 1,200,200,50 erzeugt den falschen Kreis. CIRCLE 1,200,200,50,50 zeichnet zwar den richtigen Kreis, der aber zur Ellipse verzerrt wird.

Der SCALE-Befehl kann - nebenbei bemerkt - innerhalb eines Programms oder bei ein und demselben Bildschirm - mehrfach angewendet werden. Das ergibt interessante Effekte.

SCNCLR

»SCreeN CLeaR« ist englisch und heißt etwa »Bildschirm freimachen«. Das bedeutet, daß in den Text-Modi der Speicherbereich ab $400, in den Grafik-Modi sowohl die Bit-Map als auch das Farb-RAM ab $1C00 gelöscht, beziehungsweise mit Leerzeichen oder Farbcode beschrieben werden. Welche Bereiche gelöscht werden, welche Bildschirme also, entscheidet das Argument von SCNCLR n. Die Zuordungen von »n« sind hier die gleichen, die auch beim GRAPHIC-Befehl verwendet werden.

Ein gerade sichtbarer Grafik-Bildschirm ist auch löschbar ohne Angabe eines Arguments, SCNCLR genügt in diesem Fall. Sollte im Textmodus ein Fenster mittels WINDOW definiert sein, dann bezieht sich der Löschbefehl auf den Fensterinhalt.

Bei der Anwendung des SCNCLR-Befehls in den Grafik-Modi muß bedacht werden, daß für den Multicolormodus und für den hochauflösenden Modus dieselbe Bit-Map benutzt wird. Nach SCNCLR3 ist also auch ein im Grafik-Modus 1 sinnvolles Bild verschwunden.

WIDTH

»Width« ergibt aus dem Englischen ins Deutsche übersetzt den Begriff »Breite«. Gemeint ist damit die Strichstärke, in der Zeichenbefehle auf dem Bildschirm realisiert werden.

Mit WIDTH 1 schaltet man die normale Strichdicke ein, mit WIDTH 2 wird allen grafischen Objekten die doppelte Strichbreite zugeordnet.

Alle Zeichenbefehle nach einem WIDTH-Befehl führen dann zur darin angegebenen Strichstärke.

RCLR

Dieser Befehl bildet das Gegenstück zum COLOR-Befehl. Mittels RCLR n ist es möglich, festzustellen, welche Farbzuordnungen zu den verschiedenen Quellen getroffen worden sind. »n« ist dabei eine Kennzahl mit derselben Bedeutung wie der erste Parameter der COLOR-Anweisung.

```
PRINT RCLR(2)
```

gibt also Auskunft über die Farbe, die als Multicolorfarbe 1 definiert wurde.

RGR

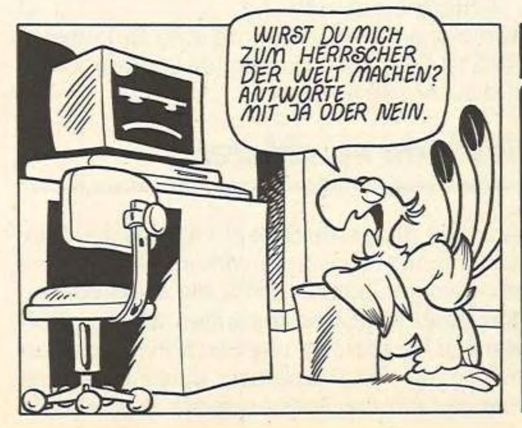
Vermutlich relativ selten benötigt, ergibt

```
PRINT RGR(x)
```

den aktuellen Grafikmodus. Das Argument x ist hier lediglich ein Dummy, hat also keine Funktion außer der eines Platzhalters. Die sich ergebende Zahl entspricht dem ersten Argument des GRAPHIC-Befehls, gibt also den Grafikmodus an.

Mit diesen Grafikbefehlen haben Sie also ein Werkzeug in der Hand, das bei richtiger Anwendung die tollsten Effekte auf dem Bildschirm ermöglicht.

(Heimo Ponnath/dm)







Hardware-Tips zum C 128

Die Hardware des C128 verbirgt noch einige Geheimnisse. Ein Teil davon wird nun offenbart. Außerdem stellen wir verschiedene Methoden vor, um mehrere Betriebssysteme im C128- oder C64-Modus in ein einziges EPROM zu brennen.

Der elektrische und mechanische Aufbau des C128 ist noch für einige Kniffe und Tricks gut. Sie können durch kleine Eingriffe an der Hardware des C128 Änderungen am System vornehmen, die zum Beispiel eine Umschaltmöglichkeit zwischen zwei verschiedenen C128-Betriebssystemen ohne großen Aufwand ermöglichen. Des weiteren lassen sich in einem einzigen 27512-EPROM vier Betriebssysteme im C128-Modus und sogar sieben im C64-Modus unterbringen.

Diesen Artikel sollten Sie aber nur lesen, wenn Sie sich mit Hardware einigermaßen gut auskennen, da Sie sonst schnell etwas an Ihrem Computer zerstören könnten. Beachten Sie bitte außerdem, daß ein Öffnen des Gerätes einen Verlust der Garantieansprüche nach sich zieht. Als Bonbon erhalten Sie noch den kompletten Schaltplan des C128. Bitte bedenken Sie, daß die vier einzelnen Teilschaltpläne als ein Ganzes angesehen werden müssen.

Übersteuerung monochromer Monitore

Ein Problem der Anpassung Commodore-fremder, einfarbiger 80-Zeichen-Monitore läßt sich leicht beheben: Es kann vorkommen, daß fremde Schwarzweiß-Monitore beim Anschluß an das monochrome Ausgangssignal (Kontakt 7 des RGB-Ausgangs) zum Übersteuern neigen. Dies macht sich dadurch bemerkbar, daß die Zeichen auf dem Bildschirm nicht mehr gestochen scharf, sondern verwaschen wirken. Der Grund liegt darin, daß das Ausgangssignal eine zu hohe Spitzenspannung aufweist, die den Eingang des Monitors übersteuert. Eine Lösung ist das Herabsetzen dieser Spitzenspannung.

Man muß nicht unbedingt durch externe Beschaltung das Signal »bändigen«. Leichter ist es, im Computer eine Änderung vorzunehmen. Dazu betrachten wir uns den dritten Teil des folgenden Schaltplans:

In der unteren Hälfte am rechten Bildrand sehen Sie die RGB-Buchse CN10. Das fragliche Signal liegt an Kontakt 7 an (Monochrome). Verfolgen Sie nun diese Leitung bis zum Transistor Q1 (2SC1815). Wenn Sie nun an die Basis des Transistors (der Kontakt, der mit dem Widerstand R26 verbunden ist) einen Widerstand von 100 Ohm anlegen und das andere Ende des Widerstandes mit Masse verbinden, haben Sie das Ausgangssignal auf ein Maß begrenzt, das Ihren angeschlossenen Monitor nun nicht mehr übersteuert.

Betriebssystem-Umschaltung eingebaut

Vielleicht ist Ihnen beim Betrachten des vierten Schaltplanes schon die merkwürdige Beschriftung der Bausteine U32 und U34 aufgefallen? Dort sind nämlich zwei verschiedene Werte für diese Bausteine angegeben. Zum einen steht da 23128, aber im gleichen Zug findet sich auch die Bezeichnung 23256. Hardware-Kenner werden nun wissen, daß ein 23128-ROM 16 KByte enthält, ein 23256 aber 32 KByte an Informationen beherbergt. Was hat es also damit auf sich?

Sehen wir uns erst einmal an, was die ROMs U32 bis U35 eigentlich enthalten. Betrachten Sie dazu auch das obere Drittel des vierten Schaltplanes:

U32 – In diesem 16-KByte-Baustein befindet sich der Basic-Interpreter für den C64-Modus und dessen Betriebssystem

U33 – Die ersten 16 KByte des Basic-7.0-Interpreters für den C128-Modus

U34 – Der Inhalt dieses ROMs besteht aus den zweiten 16 KByte des Basic-7.0-Interpreters

U35 – Dieser ROM-Baustein enthält 16 KByte Kernel (Betriebssystem) für den C128-Modus

Wir stellen also fest: vier 16-KByte-Bausteine. Warum können aber U32 und U34 mit 32-KByte-Bausteinen bestückt werden?

Es ist durchaus möglich, die Inhalte der ROMs U32 und U35 in ein einziges EPROM vom Typ 27256 zu brennen, in den Platz U32 zu stecken und den Steckplatz U35 freizulassen. Gleiches geschieht dann auch mit den Inhalten der ROMs U33 und U34, die, in ein EPROM 27256 gebrannt, im Steckplatz U34 ein neues Zuhause finden. Die Auswahl, ob nun vier Bausteine zu je 16 KByte oder zwei zu je 32 KByte angesprochen werden, erfolgt über einen dreifachen Ein-/Aus-Schalter. Betrachten wir uns den Plan 2. Im unteren rechten Viertel finden Sie die PLA 8721 (U11), die für die Selektion der einzelnen Bausteine verantwortlich ist.

Am Pin 27 können Sie eine Brücke (J3) erkennen, die für die Auswahl der verwendeten Typen zuständig ist. Bei offener Brücke spricht die PLA die vier 16-KByte-Bausteine an, bei geschlossener die 32 KByte-EPROMs. Des weiteren müssen wir noch den Plan 4 betrachten. Jeweils links oberhalb der ROMs U32 und U34 finden Sie eine Brücke (J4 und J6), die ebenfalls eine Rolle bei der Art der verwendeten Bausteine spielen. Sind diese Brücken offen, müssen 16-KByte-Bausteine eingesetzt sein, bei geschlossenen Brücken wären es die 32 KByte-EPROMs. Was kann man nun aber damit anstellen?

Eine Möglichkeit wäre, in einem 27256-EPROM in den unteren 16 KByte den Inhalt des ROMs U32 und im oberen 16-KByte-Bereich ein geändertes Kernel für den C128-Modus einzuprogrammieren. Würde nun der Baustein U35 in seinem Sockel belassen und das neue EPROM in den Platz U32 eingesetzt, so könnte ohne viel Aufwand intern zwischen zwei C128-Betriebssystemen umgeschaltet werden. Es müßten nur die Brücken J3, J4 und J6 mit einem dreifachen Ein-/Aus-Schalter gleichzeitig überbrückt oder geöffnet werden. Allerdings ist zu beachten, daß sich im Steckplatz U34 ebenfalls ein 32 KByte-EPROM befinden muß, das im oberen 16-KByte-Bereich zum Beispiel einen geänderten Teil des Basic-7.0-Interpreters enthält.

Wie Sie nun, wie eingangs erwähnt, mehrere Betriebssysteme in einem 27512-EPROM unterbringen können, erklären wir Ihnen ein paar Absätze später.

Bildschirm »einfrieren«

Vielleicht möchten auch Sie gerne Bildschirmfotos machen, die »lupenrein« und nicht verwaschen wirken. Oder sei es nur, um eine besonders gelungene Grafik, die sich leider zu schnell bewegt, im Detail betrachten zu können. Im alten C64 konnte man als einzige Möglichkeit das Bild anhalten, wenn man den READY-Eingang des Prozessors durch ein Signal belegte. Nahm man jedoch die Sperre wieder weg, stürzte

das Programm jedes zweite Mal ab. Im C 128 läßt sich ein stehendes Bild auch ohne das Risiko eines Systemabsturzes erreichen. Dazu muß nur das Signal »DMARQST«, das von Pin 11 des VIC (U21) kontrolliert wird, auf HIGH gelegt werden. Dies läßt sich durch einen einfachen Ein-Taster bewerkstelligen. Das eine Kabel verbinden Sie mit Pin 11 des VIC, das andere mit +5V. Schon erhalten Sie ein stehendes Bild, das nach Loslassen des Knopfes in 99,9 Prozent aller Fälle weiterläuft. Den Schalter können Sie sich zum Beispiel auf der rechten Seite neben dem Reset-Knopf einbauen und das Kabel frei unter der Platine verlegen.

Farbabgleich

Sollten Ihnen die Farben, die Ihr C 128 liefert, etwas schwach und ausgebleicht vorkommen, kann es sein, daß der Video-Prozessor nicht die richtige Taktfrequenz erhält, die nötig ist, um ihn mit dem Fernseher (Monitor) zu synchronisieren. Abhilfe kann ein Nachregulieren der Frequenz sein. Dazu drehen Sie den Drehkondensator C20 (Plan 3, oben rechts – Sitz auf der Platine neben dem VIC) etwas nach links oder rechts. Probieren Sie so lange, bis Sie den besten Farbwert erhalten.

C 64-Modus gleich beim Einschalten

Da derzeit noch nicht allzuviel Software für den C 128- und CP/M-Modus auf dem Markt erhältlich ist, wird der C 128 wohl meistens im C 64-Modus betrieben. Es ist aber mit der Zeit lästig, immer beim Einschalten oder einem Reset die Commodore-Taste gedrückt zu halten oder mit dem Befehl GO 64 in dessen Modus zu gelangen. Abhilfe kann die aus Bild 1 ersichtliche Schaltung schaffen.

Das Funktionsprinzip: Der C 128 springt bei eingestecktem Modul im Expansion-Port automatisch in den C 64-Modus. Es muß also nur das Vorhandensein eines Moduls simuliert werden. Da das System auf Grund des Pegels der EXROM- oder GAME-Leitung am Expansion-Port feststellt, ob ein Modul eingesteckt ist, muß nur die EXROM-Leitung auf Masse gelegt werden. Verbindet man diese Leitung nun direkt mit Masse, so springt der Computer in die C 64-Betriebsart. Es folgt aber eine enttäuschende Einschaltmeldung: »30719 BASIC BYTES FREE« steht da auf dem Bildschirm. Was ist der Grund, daß plötzlich 8 KByte weniger zur Verfügung stehen? Für den Computer steckt nun ein Modul am Expansion-Port, obwohl dort keines zu sehen ist. Auch ein Blick in den Speicherbereich ab $8000 zeigt, daß dieser Bereich nicht ansprechbar ist. Also muß dieser Bereich wieder durch Entfernen der Modulkennung dem Gesamtspeicher zugänglich gemacht werden. Eine Möglichkeit dazu ist, einen Kondensator an entsprechender Stelle einzubauen.

Bei genauer Betrachtung der Funktionsweise eines Kondensators zeigt sich, daß dieser im ersten Moment der Aufladungsphase einen Kurzschluß (geschlossener Schalter) darstellt. Damit wäre eine automatische Modulkennung durch Beschalten des EXROM-Anschlusses mit Masse erreicht. Bei ersten Versuchen mit einem 10µF-Kondensator funktionierte die Sache bereits recht gut. Das System meldete sich zwar immer noch mit »30719 BASIC BYTES FREE«, doch war der Speicherbereich ab $8000 nun frei. Nach weiteren Experimenten mit verschiedenen Kapazitäten stellte sich dann heraus, daß sich bei 3,3µF der Kondensator zum Zeitpunkt der Modulkennung genügend entladen hat. Zum Zeitpunkt der Berechnung der Speichergröße aber ist er genügend aufgeladen, um die Berechnungsroutine nicht weiter zu stören.

Bei geschlossenem Schalter S1 meldet sich nun der C 128 direkt mit der Einschaltmeldung des C 64 und »38911 BASIC BYTES FREE«.

Dieser Kondensator muß aber schaltbar gemacht werden, da man sonst nie mehr in den C 128-Modus gelangen kann.

Mehrere Betriebssysteme in größeren EPROMs

Für den C 128-Modus:

Mit der in Bild 2 abgebildeten Schaltung wird es nun möglich, auf einfache Weise vier Betriebssysteme im C 128-Modus benutzen zu können (das Betriebssystem ist im ROM U35 enthalten). Diese Schaltung läßt sich ebenfalls verwenden, wenn man einen Teil des Basic-7.0-Interpreters »umstricken« möchte (zum Beispiel, um die Einschaltmeldung, die sich im ROM U34 befindet, zu verändern. Es kann aber auch eine OLD-Routine integriert werden. Eine solche Routine, die sich zum Einbinden eignet, wurde in Ausgabe 12/85 veröffentlicht).

Ein EPROM vom Typ 27512 kann maximal 64 KByte an Informationen aufnehmen. Das Betriebssystem des C 128 umfaßt aber »nur« 16 KByte. Es ist also möglich, vier C 128-Betriebssysteme durch Umschalten zur Verfügung zu haben.

Zum Einbau gehen Sie folgendermaßen vor: Nachdem Sie Ihre verschiedenen Betriebssysteme (oder Interpreter-Versionen) getestet und nacheinander in ein 27512-EPROM gebrannt haben, öffnen Sie Ihren C 128 (Vorsicht: Garantieverlust!).

Biegen Sie Pin 1 und 27 des EPROMs nach oben, so daß sie keinen Kontakt mehr mit dem Computer haben können.

UMSCHALTUNG PC-128 / C-64 RESET

\+ C1
S1
22 9 1
EXPANSION PORT
Z A
C1 = ELKO 3.3 µF
S1 = EIN = C-64 RESET
S1 = AUS = PC-128 RESET

Bild 1. Mit dieser Beschaltung springt der C 128 nach dem Einschalten sofort in den C 64-Modus

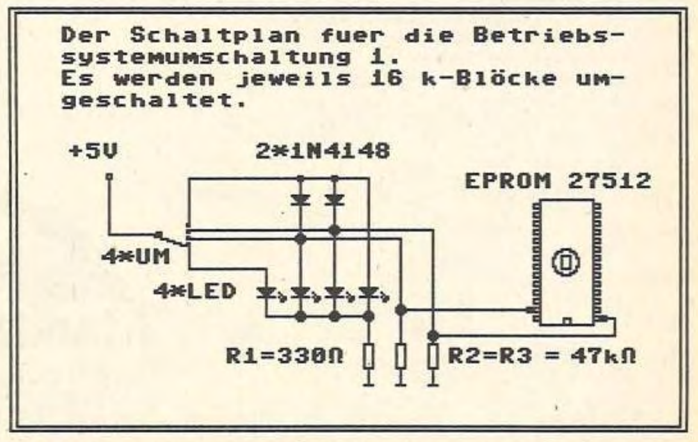
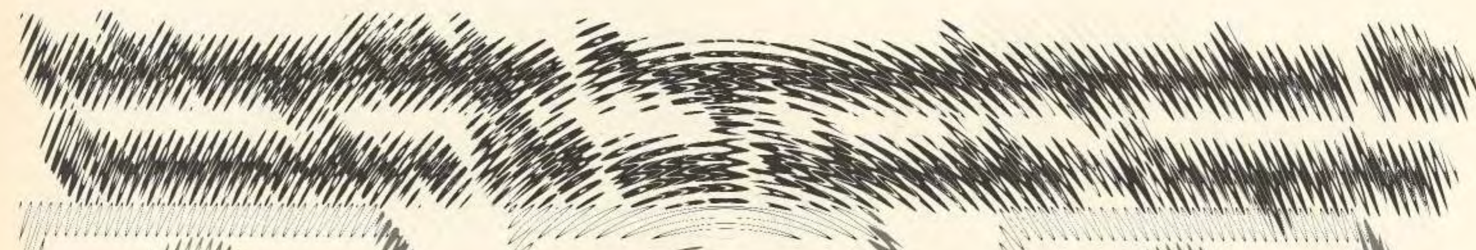
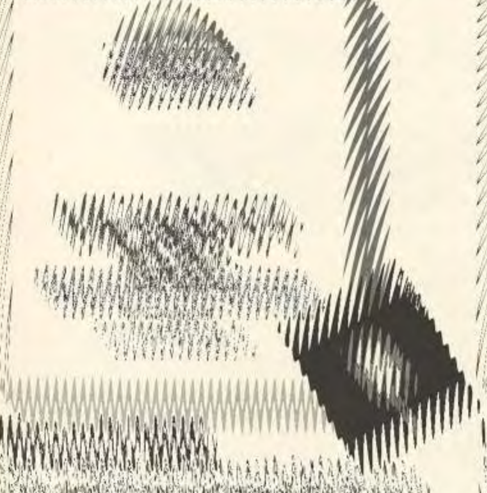


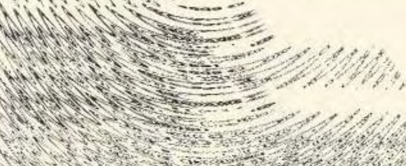
Bild 2. Vier verschiedene C 128-Betriebssysteme in einem EPROM

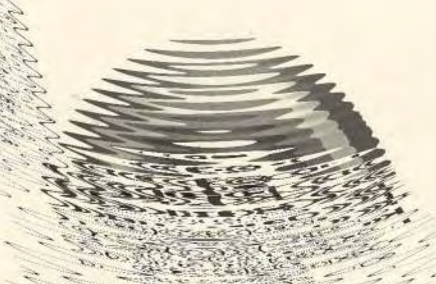


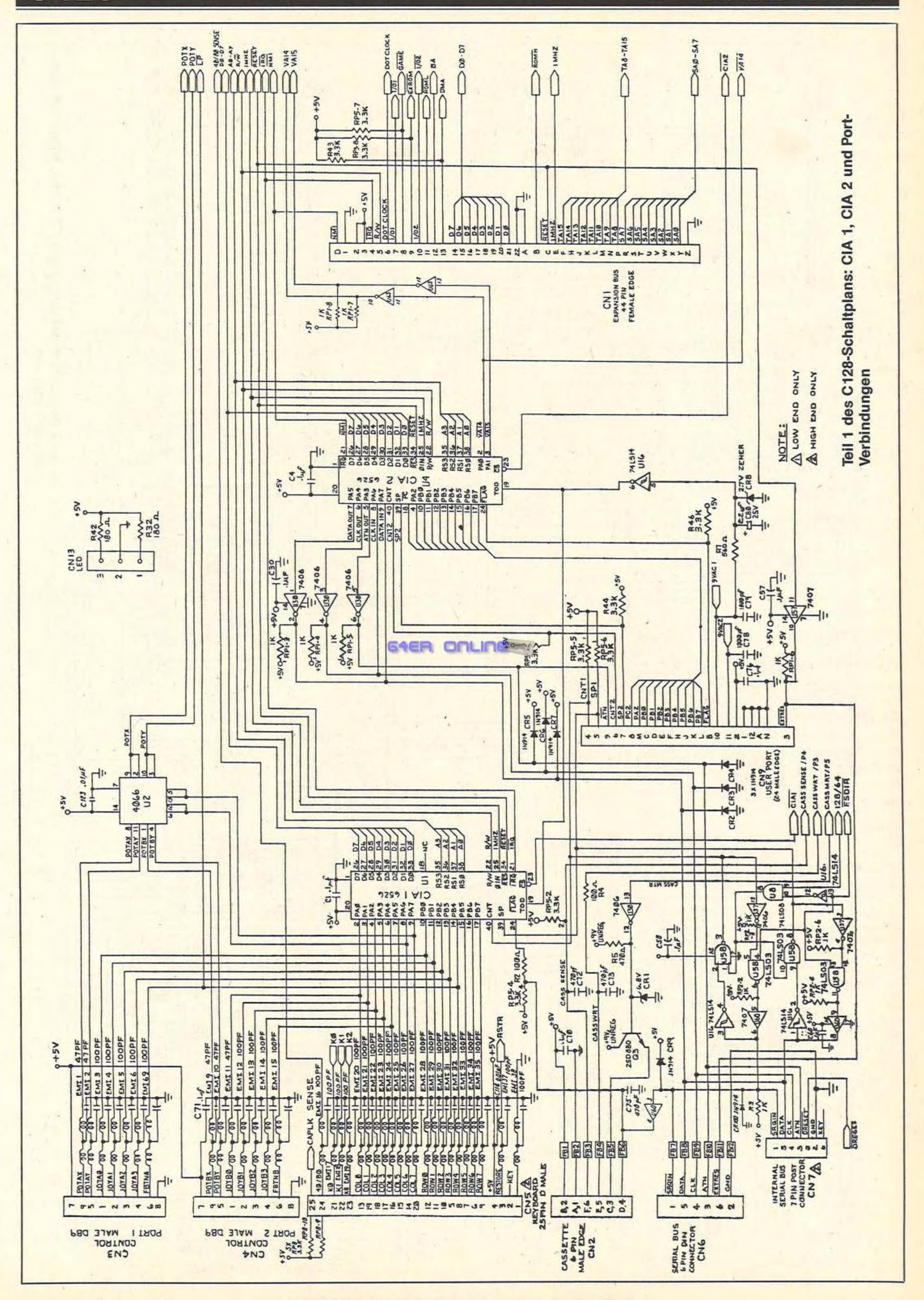




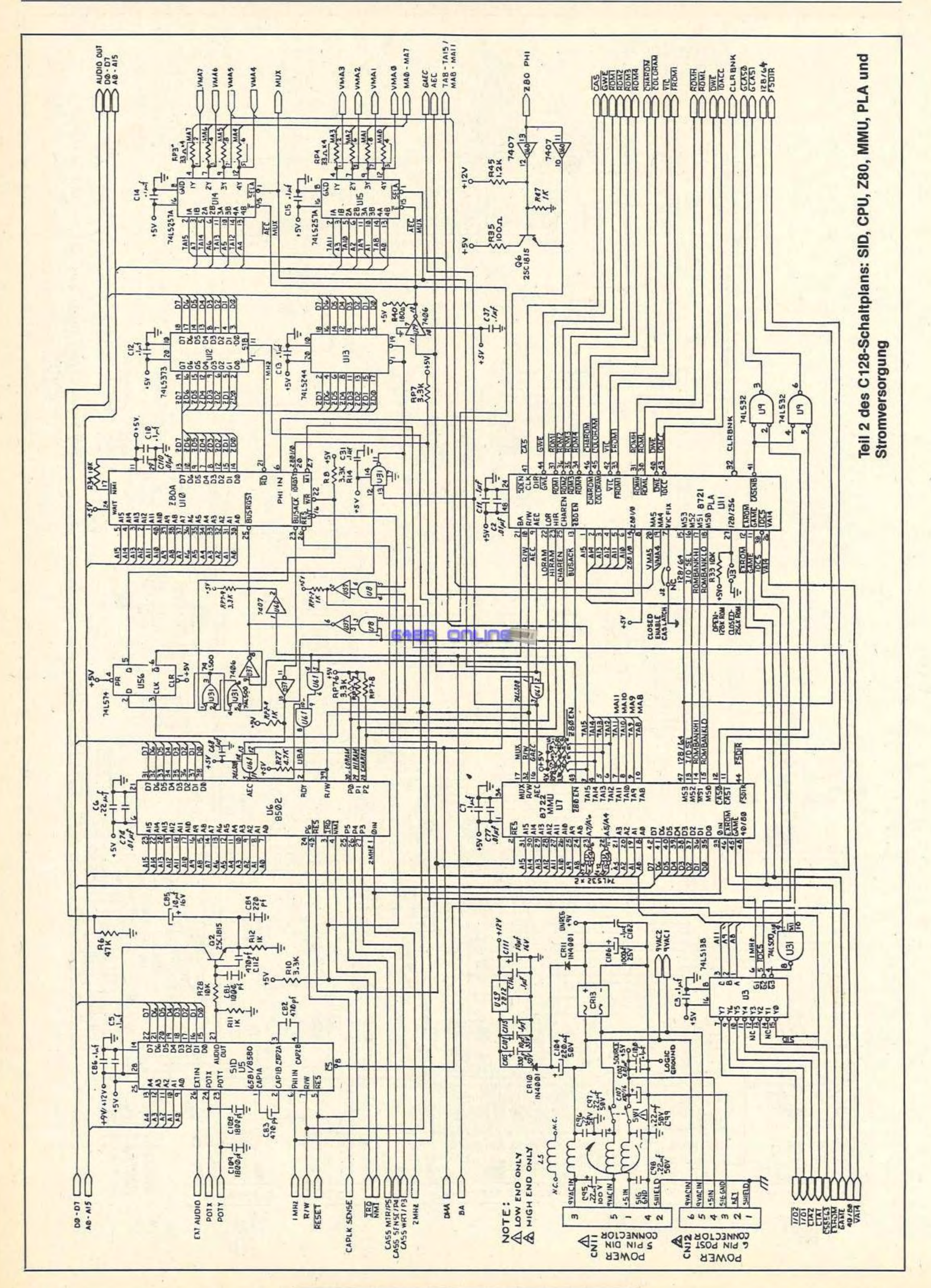




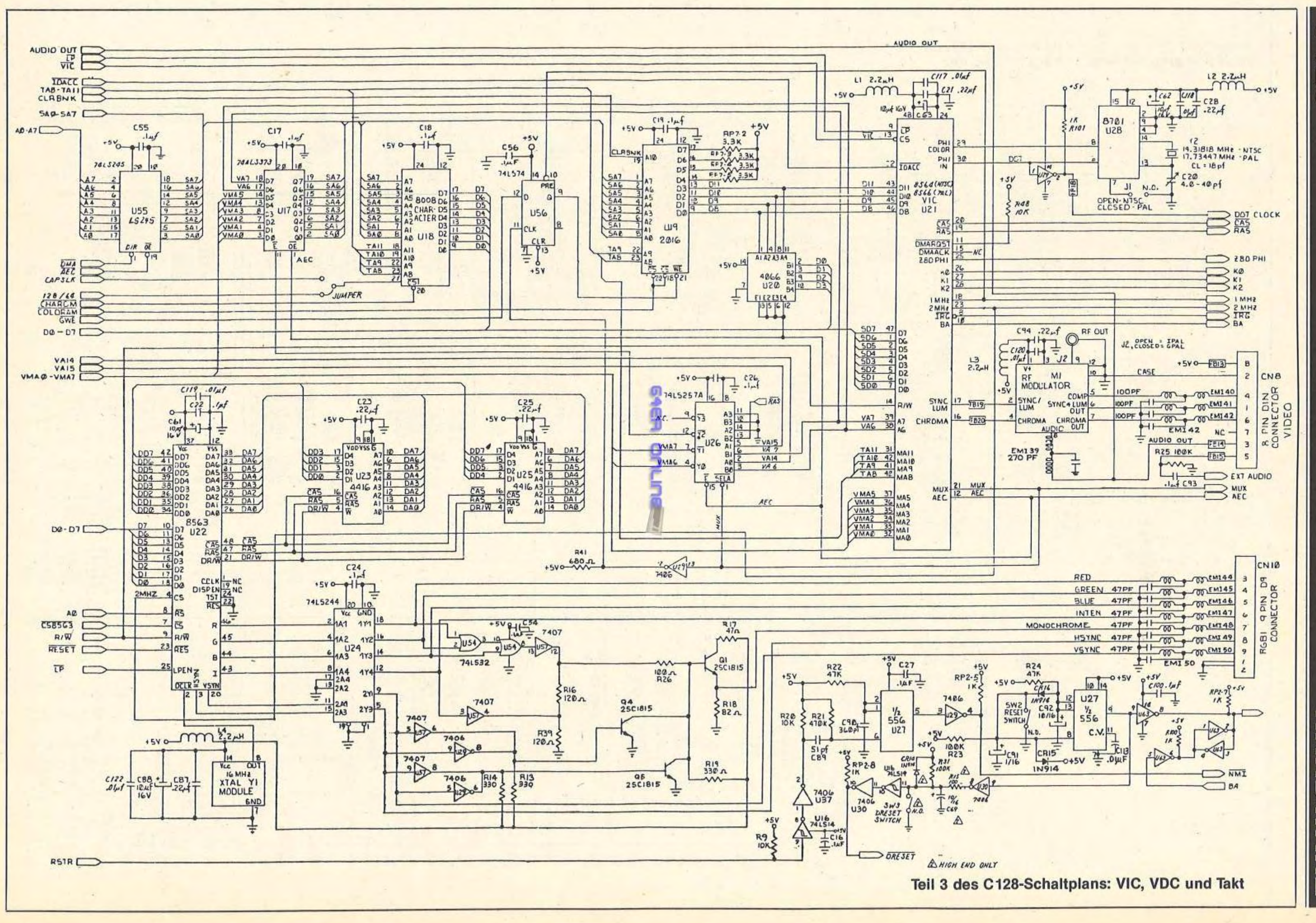




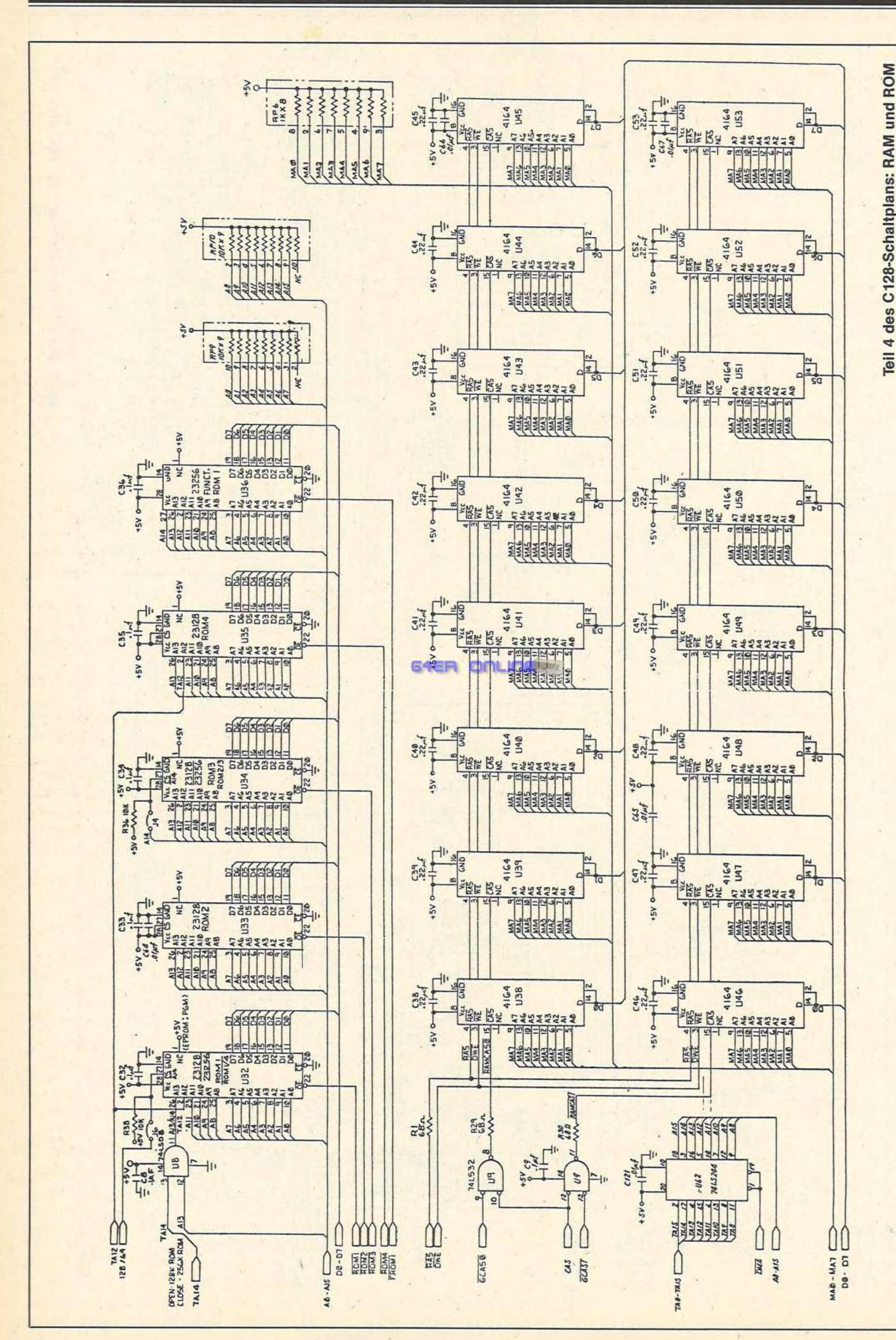
Teil 1 des C128-Schaltplans: CIA 1, CIA 2 und Port-Verbindungen



Teil 2 des C128-Schaltplans: SID, CPU, Z80, MMU, PLA und Stromversorgung



Teil 3 des C128-Schaltplans: VIC, VDC und Takt



Teil 4 des C128-Schaltplans: RAM und ROM

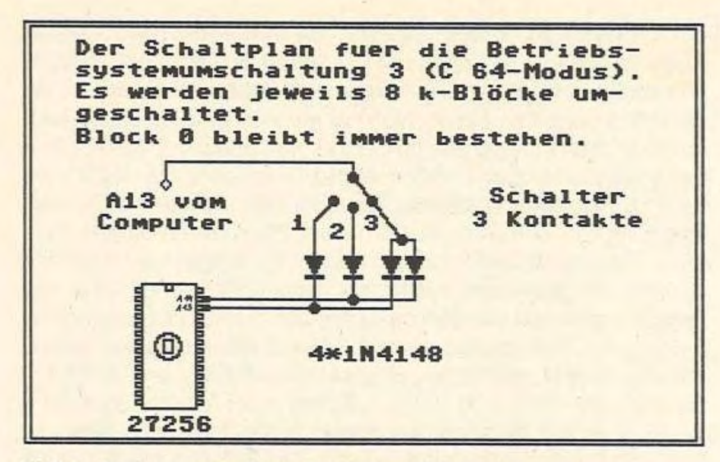


Bild 3. Drei verschiedene Betriebssysteme für den C64-Modus in einem EPROM

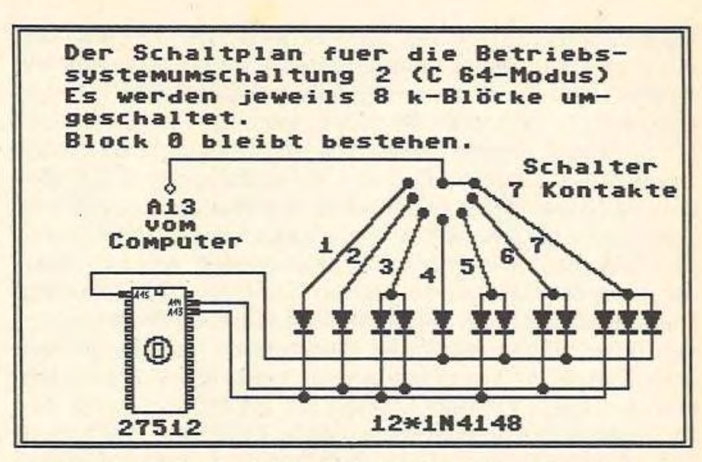


Bild 4. Mit dieser Schaltung lassen sich in einem EPROM sieben verschiedene C64-Betriebssysteme im C128 unterbringen

Stecken Sie das EPROM in den jeweiligen Sockel (Kernel: U35, Interpreter Low: U33, Interpreter High: U34) und beschalten Sie die beiden herausragenden Beinchen wie aus Bild 2 ersichtlich.

Die Schaltung ist nicht absturzfrei. Je nachdem, ob die gerade bearbeitete Routine des jeweiligen Systems mit der originalen übereinstimmt, kann es auch ohne Absturz funktionieren. Das System stürzt nur dann ab, wenn innerhalb verschiedener Routinen umgeschaltet wird.

Für den C64-Modus:

Hier stellen wir Ihnen zwei Arten der Umschaltung vor: zum einen für drei, zum zweiten für sieben verschiedene Kernel-Versionen. Beginnen wir mit der kleineren der beiden Möglichkeiten.

Hierfür benötigen Sie ein EPROM des Typs 27256 (32 KByte). Zu beachten ist hierbei, daß der Interpreter-Bereich ($A000 bis $BFFF) in den untersten 8 KByte des EPROMs liegen muß. Die anderen 24 KByte können Sie mit drei verschiedenen Kernel-Versionen für den C64-Modus belegen. Nachdem das EPROM gebrannt ist, schrauben Sie den Computer auf. Der betreffende Steckplatz trägt die Bezeichnung U32.

Entnehmen Sie nun den Original-ROM-Baustein. Biegen Sie Pin 26 und 27 des 27256-EPROMs in die Waagerechte, so daß diese keinen Kontakt mehr mit dem Sockel haben können und setzen Sie das EPROM in den Steckplatz U32 ein. Löten Sie ein Kabel an die Unterseite der Platine, und zwar an den Pin 26 des Steckplatzes U32. Dieses Kabel verbinden Sie mit dem Mittelkontakt des 3fach-Umschalters. Beschalten Sie nun die beiden freien IC-Beinchen und den Umschalter so, wie es aus Bild 3 ersichtlich ist.

Diese Art der Umschaltung ist nicht absturzfrei.

In einem EPROM vom Typ 27512 lassen sich, abzüglich des Interpreters, der die ersten 8 KByte des EPROMs belegen muß, sieben verschiedene Betriebssysteme unterbringen. Denkbar wären zum Beispiel das 64'er-DOS, Speeddos, Hypra-Kernel, das Original-Betriebssystem und ein paar auf den Eigenbedarf abgestimmte Varianten.

Nachdem das EPROM gebrannt wurde, öffnen Sie den Computer und entfernen den Baustein U32. Biegen Sie Pin 1, 26 und 27 des EPROMs in die Waagrechte und stecken das 64 KByte-EPROM in den Steckplatz U32.

Löten Sie nun an der Unterseite der Platine einen Draht an den Pin 26 des Bausteins U32. Dieser Draht wird mit dem Mittelkontakt des 7fach-Schalters verbunden. Die Beschaltung der frei in der Luft hängenden IC-Beinchen mit dem Drehschalter entnehmen Sie bitte Bild 4.

Diese Möglichkeit der Umschaltung ist nicht absturzfrei.

Wenn Sie auch etwas zum Thema »Tips und Tricks zur C128-Hardware« wissen, schreiben Sie uns doch. Wir werden alle Anregungen prüfen und gegebenenfalls auch veröffentlichen. (Franz Sauer/dm)

Was ist CP/M?

Noch immer gibt CP/M vielen C128-Anwendern Rätsel auf. Was ist CP/M und wie geht man damit um? Wir stoßen Ihnen ein Tor in die CP/M-Welt auf.

Die Entwicklung der Mikrocomputertechnik steckte Anfang der siebziger Jahre noch in den Kinderschuhen. Die neueste Errungenschaft war damals der Intel-8080-Prozessor, der Vorgänger des legendären Z80 (Zilog 80), der auch im C128 eingebaut ist. Es entstand bald der Wunsch für alle Computer, die mit dem 8080-Prozessor ausgerüstet waren, ein einheitliches Betriebssystem zu schaffen. In dieser Zeit waren Computer nur zu Preisen erhältlich, die uns heute utopisch erscheinen. Die erste CP/M-Version 1.4 kam 1975 auf den Markt, fand aber damals wenig Beachtung. 1979/80 brachte Digital Research die erweiterte CP/M-Version 2.2 heraus. CP/M war jetzt zu einem universellen Betriebssystem geworden, das sehr schnell Anklang fand. Zwar gilt auch heute noch CP/M 2.2 als Standard, doch wurde CP/M zur Version 3.0 weiterentwickelt, die zur Version 2.2 voll kompatibel ist.

Der Anwender findet beim ersten Kontakt mit CP/M zwar eine Menge Programme auf der Systemdiskette, versucht aber vergeblich, diese mit dem vielleicht gewohnten »RUN«-Befehl zu starten. Spätestens an diesem Punkt fällt auf, daß CP/M mit dem bisher bekannten C64- und C128-Modus wenig gemeinsam hat. Wo liegen die Gründe für dieses Phänomen? Wie Sie wissen, befindet sich der C128 nach dem Einschalten automatisch im Basic-Modus. Hier können Sie nach Belieben mit dem eingebauten Basic-Interpreter arbeiten. Dieser Basic-Interpreter ist der Teil, der entweder im Dialog oder im Programm Befehle entgegennimmt und ausführt. Unter CP/M steht ein derartiger Systembaustein in der Grundausstattung nicht zur Verfügung. Einzig und allein durch die Möglichkeit, Unmengen von professioneller Soft-

ware einsetzen zu können, eröffnen sich dem Anwender bisher ungeahnte Dimensionen. Wären Sie beispielsweise mit Ihrem C 128 auf einer einsamen Insel (Stromanschluß vorausgesetzt), könnten Sie ohne weiteres im C64- und C128-Modus eigene Programme erstellen, nicht so mit der CP/M-Systemdiskette. Das im Lieferumfang des C128 enthaltene Systempaket enthält nicht einmal den eigentlich vorgesehenen Assembler und Debugger. Natürlich können diese Bestandteile von CP/M nachbestellt werden. Allerdings werden dann wieder Kosten in Höhe von 80 Mark fällig. Dafür erhält man aber zwei Disketten mit vielen Programmen, den notwendigen Assembler-Werkzeugen und ein mehrere 100 Seiten umfassendes englischsprachiges Handbuch. Erst in diesem Moment können Sie die Möglichkeiten des Prozessors und des Betriebssystems für eigene Programme nutzen, falls Sie den 8080-Maschinencode beherrschen.

Programmieren unter CP/M

Eine der Stärken von CP/M liegt eindeutig in der Vielzahl von Programmiersprachen, die für dieses Betriebssystem angeboten werden. Von Fortran, Cobol, Algol, Pascal und C über Basic und Pilot bis hin zu Lisp sind die verschiedensten Hochsprachen mit deren speziellen Anwendungsgebieten unter CP/M verfügbar. Dabei handelt es sich zum Großteil um sogenannte Compilersprachen. Ein Compiler übersetzt den Programmtext in einem oder mehreren Durchgängen in direkt ausführbare Programmdateien. Den Programmtext, der entweder mit dem Systemeditor »ED« oder einem Textverarbeitungsprogramm (zum Beispiel Wordstar) erstellt wurde, nennt man auch Quellcode. Entsprechend bezeichnet man das ablauffähige Programm als Objektcode.

Bei einigen dieser Compiler, speziell unter Basic, gehört ein Interpreter zum Lieferumfang. Der Interpreter prüft und übersetzt den Programmtext während des Programmablaufs und führt die als reservierte Standardworte erkannten Befehle und Funktionen aus. Der Programmtext entspricht dabei jedoch nicht einem direkt ablauffähigen Maschinenprogramm, sondern muß immer aufs neue interpretiert werden. Die Vorteile des Interpreters liegen in den schnellen Korrektur- und Testmöglichkeiten. So ist es beispielsweise möglich, eine Variable, einen Befehl oder eine Funktion nachträglich einzufügen, zu löschen oder zu verändern und das Programm danach sofort zu starten. Wird ein Compiler verwendet, muß das Programm nach jeder Änderung neu compiliert werden. Dies ist zwar im Hinblick auf Test und Korrektur sehr zeitaufwendig, das Ergebnis ist allerdings von beeindruckender Geschwindigkeit.

Die CP/M-Organe

CP/M ist aus drei »lebenserhaltenden« Grundelementen aufgebaut. Diese Elemente werden für den Anwender erst interessant, wenn er sich näher mit dem System befaßt oder gar in die Assemblerprogrammierung einsteigen will. Zum ersten wäre hier das BDOS zu nennen. Dieser Systembestandteil verrichtet die Hauptarbeit. Ohne den kleinen Bruder des BDOS, das sogenannte BIOS, läuft aber trotzdem nicht viel. Das BDOS regelt den Datenfluß zwischen dem Anwenderprogramm und den verschiedenen Aus- und Eingabegeräten. Angesprochen wird das BDOS über eine Vielzahl von Nummern, die die einzelnen Routinen repräsentieren. Vom BDOS werden die Daten an das BIOS übergeben, das endgültig den Datentransfer in die Wege leitet. Das BIOS greift auf einige Tabellen im Speicher zurück, in denen die wichtigsten Informationen über den momentanen Systemstatus gespeichert sind. Der erfahrene Assemblerprogrammierer hat jederzeit die Möglichkeit, diese Tabellen zu verändern und seinen Anforderungen anzupassen. Allerdings müssen solche Manipulationen mit größter Umsicht durchgeführt werden, da es sehr schnell zu einem nicht eingeplanten Systemabsturz kommen kann. Der Dritte im Bunde schließlich ist der »CCP«, der sogenannte Control-Command-Prozessor. Der »CCP« ist für Ihre Eingaben zuständig. Er stellt unter anderem fest, ob Ihre Eingabe einen residenten oder transienten Befehl darstellt. Residente Befehle sind fest in das eingeladene CP/M-System eingebunden, während transiente immer erst von Diskette geladen werden müssen, bevor sie zur Ausführung gelangen. Unter die residenten Befehle fallen beispielsweise »A:...E:«, »DIR«, »ERASE«, »RENAME«, »TYPE« und »USER«. Die Befehle »DIR« und »TYPE« dürfen nicht mit Optionen versehen werden, da sonst automatisch das transiente Äquivalent von Diskette eingeladen wird. Ein Beispiel hierfür wäre der Befehl »DIR«. Verwendet man »DIR [FULL]«, so lädt der C128 das Programm »DIR.COM« ein und arbeitet die entsprechende Option ab. Probleme treten dann gerne auf, wenn man nur mit einem Laufwerk arbeitet. In diesem Fall könnte der C128 auf einer Arbeitsdiskette ohne »DIR.COM« den Befehl nicht ausführen. Man kann sich jedoch behelfen, indem man dem Befehl entsprechende Option »E:« voranstellt (in diesem Fall also »E:DIR[FULL]«). Hierbei wird das sogenannte virtuelle Laufwerk angesprochen. Anstelle von Lese- und Schreibzugriffen auf eine tatsächliche Floppystation werden diese Operationen lediglich im »RAM« ausgeführt. Das System gibt anschließend eine Aufforderung zum Einlegen der entsprechenden Diskette aus. CP/M 3.0 unterstützt neben dem virtuellen Laufwerk noch vier weitere hardwaremäßige Laufwerke (A: bis D:). und M: (Speichererweiterung z.B. 512kbyte)

Ausgereifte Software

Aufgrund der langjährigen Erfahrungen, die teilweise mit älteren CP/M-Versionen gemacht wurden, reiften früh entstandene Softwarepakete, wie etwa Wordstar 3.0, dBase II und Multiplan weitgehend aus. Sowohl Programm- als auch CP/M-Versionen wurden fortlaufend erweitert und verbessert. Dies kommt dem heutigen Anwender vollständig zugute. Er kann gewaltige Programmpakete nutzen, die sich schon seit einigen Jahren großer Beliebtheit erfreuen und auch unter MS-DOS auf IBM-Computern und kompatiblen Geräten verfügbar sind.

Public-Domain-Software

Wie kaum unter anderen Betriebssystemen, hat sich unter CP/M und MS-DOS (einem erweiterten Äquivalent für IBM-Computer und kompatible Geräte) im Laufe der Jahre ein großes Kontingent an sogenannter Public-Domain-Software herausgebildet. Das Besondere an dieser Art von Software ist, daß sie von Anwendern für Anwender geschrieben wurde. Es fehlt hier völlig die sonst übliche kommerzielle Vermarktung mit Preisen für den großen Geldbeutel. Hauptsächlich in den Vereinigten Staaten von Amerika erhält man Public-Domain-Software unter CP/M, sei dies nun in Form des sogenannten »Download« bei speziellen Mailboxen, über User-Groups oder auch über Softwarehändler, die häufig sehr viele dieser Public-Domain-Programme »in Bausch und Bogen« vertreiben. Das Terminal-Programm »KERMIT« hat sich beispielsweise als Public-Domain-Programm einen Namen gemacht. Es besteht die Hoffnung, daß sich das Gebiet der Public-Domain-Software nach der Renaissance des Betriebssystems CP/M durch den Commodore 128 und die CP/M-fähigen Schneider-Computer auch in Deutschland durchsetzen wird.

(bj/rf)

Tips & Tricks zu CP/M

Hier finden Sie tolle Neuigkeiten und eine Menge Anregungen, mit denen Sie CP/M in den Griff bekommen und dessen Vorteile voll ausnutzen können.

Nachdem man sich als CP/M-Einsteiger durch das sehr dürftig ausgefallene Handbuch von Commodore gearbeitet hat, wird zwar klar, daß es sich hier um ein Betriebssystem handeln muß, welches mit dem C 128 und C 64 nichts zu tun hat. Es bleiben für den Einsteiger wie auch für den Profi viele Fragen offen. »Tips & Tricks zu CP/M« zeigt weiterführende Anwendungsmöglichkeiten von CP/M auf, die sich nicht auf das Kopieren von Dateien mit »PIP« beschränken.

Submit – oder so ähnlich...

Im Handbuch stößt man irgendwann einmal darauf. »SUBMIT.COM« und »PROFILE.SUB« werden dort in der gewohnt mageren Art und Weise beschrieben. Die eigentliche Arbeits- und Wirkungsweise bleibt im Dunkeln verborgen. Also werden wir etwas Licht ins »SUBMIT«-Dunkel bringen. »SUBMIT« ist ein Programm auf der CP/M-Systemdiskette, mit dem mehrere »COM«-Files nacheinander und ohne Unterbrechung abgearbeitet werden können. Natürlich muß dem Computer irgendwie mitgeteilt werden, welche Programme er in welcher Reihenfolge abarbeiten soll. Da man am besten an einem Beispiel lernt und versteht, betrachten wir eine mögliche Anwendung von »SUBMIT« etwas genauer.

Man stelle sich den für einen CP/M-Programmierer klassischen Fall vor: Ein Programm wird im Quellcode eingetippt, compiliert, eventuell assembliert und gelinkt. Es wäre eine echte Arbeitserleichterung, wenn man nicht jedesmal die einzelnen Programme aufrufen müßte, sondern diese automatisch nacheinander geladen würden. »SUBMIT« bietet genau diese Möglichkeiten. Nehmen wir an, Sie haben den Quellcode eines Programms in einer Datei namens »PRG.C«. Bei der letzten Compilation tauchte ein Fehler auf und es soll neu editiert und umgewandelt werden. Um sich nun das einzelne Starten der Compiler-Teile zu sparen, erstellt man als erstes die »SUBMIT«-Datei, entweder mit dem CP/M-Editor oder einer Textverarbeitung. Der erstellte Text sieht folgendermaßen aus:

```
ED PRG.C
CC PRG
MAC PRG
LNK PRG
```

Diese Datei erhält den Namen »COMPILE.SUB«. Das Kürzel »SUB« ist für die Arbeit mit »SUBMIT« zwingend notwendig. Das Ganze wird mit »SUBMIT COMPILE« gestartet. Als erstes wird der Editor (ED) mit der Datei PRG.C geladen. Nachdem die Änderung durchgeführt und der Editor nach dem Speichern wieder verlassen wird, tritt automatisch der Compiler (CC) in Aktion und compiliert die Quelldatei PRG.C. In derselben Weise agieren im Anschluß der Assembler (MAC) und der Linker (LNK). Wohlgemerkt, alle Programme werden automatisch gestartet, gesteuert durch »SUBMIT«. Natürlich müssen sich alle angesprochenen Programme und Dateien auf derselben Diskette befinden.

Eine weitere Anwendung wäre die komplette Steuerung von zusammenhängenden Programmabläufen. Für umfangreiche Anwendungen sind meist mehrere Programme notwendig, um das gewünschte Ergebnis zu erhalten. Ein Beispiel wäre die Lagerverwaltung. Man benötigt Programme zur Datenerfassung, Datenauswertung und Datenaktualisierung. Da diese Programme, um Fehler zu vermeiden, nicht wahllos durcheinander gestartet werden dürfen, wird der genaue Programmablauf in einer »SUB«-Datei festgehalten und mit »SUBMIT« gestartet.

Doch bleiben wir vorerst bei unserem Compiler-Beispiel. Was nützt es, wenn die Programmaufrufe nicht variabel gehalten werden können, wenn man für jedes neue Programm eine eigene »SUB«-Datei erstellen muß? Natürlich bietet »SUBMIT« diesen Komfort. Es sind bis zu neun Variable möglich. Zugelassen ist wieder jedes ablauffähige COM-File. Die Variable werden in einer besonderen Form angegeben: als erstes ein Dollarzeichen und dann die Nummer der jeweiligen Variable. Die zu übergebenden Werte werden im »SUBMIT«-Aufruf mit angegeben. Dazu wieder ein kleines Beispiel. Als erstes sollen alle Dateien angezeigt werden, die Assembler-Quellcode enthalten (Dateikennung: ASM). Danach wird die angegebene Datei assembliert und gelinkt. Ist dies erfolgreich abgeschlossen, werden die überflüssigen, aus der Assemblierung entstandenen Dateien mit geändertem Namen auf ein anderes Laufwerk kopiert. Auf der Originaldiskette werden diese Dateien (Dateienkennung: SYM,PRN) gelöscht und das Inhaltsverzeichnis der Diskette auf den Bildschirm gebracht. Um diese bereits relativ umfangreiche Aktion mit »SUBMIT« auszuführen, wird folgende SUB-Datei benötigt:

```
DIR *.ASM
MAC $1
HEXCOM $1
PIP
<B:$2.SYM = A:$1.SYM
<B:$2.PRN = A:$1.PRN
<
ERA $1.PRN
ERA $1.SYM
DIR
```

Um die allgemeine Verwirrung nicht noch zu steigern, folgt eine ausführliche Erklärung der obigen Datei. Die Datei wird mit dem Namen »TEST.SUB« gespeichert. Vergewissern Sie sich immer vor einem »SUBMIT«-Aufruf, daß alle angeforderten Programme auf der entsprechenden Diskette vorhanden sind. Nehmen wir an, das Assemblerprogramm »DISK.ASM« soll assembliert werden und die zu kopierenden Dateien sollen die Namen »DRUCK.SYM« und »DRUCK.PRN« erhalten. Um das gewünschte Ergebnis zu erhalten, wird »SUBMIT« folgendermaßen aufgerufen:

```
SUBMIT TEST DISK DRUCK
```

Der erste Parameter nach »SUBMIT« gibt die auszuführende SUB-Datei an. Alle weiteren Parameter werden den in der SUB-Datei verwendeten Variablen zugeordnet. So erhält im Beispiel $1 den Wert DISK und $2 den Wert DRUCK. Tippfehler in bezug auf die Variablen nimmt »SUBMIT« sofort übel. Noch vor der Ausführung wird die Prozedur auf Ihre Syntax überprüft. Leider überprüft »SUBMIT« nicht, ob alle verlangten Dateien und Programme auf den entsprechenden Laufwerken zur Verfügung stehen. Hier wären wir bei einem kleinen Manko angelangt. Es gibt keine Möglichkeit, eventuelle durch Falscheingaben auftretende Fehler abzufangen. Wenn beispielsweise statt »DISK« versehentlich »DIS« eingegeben wird, werden alle Programme der SUB-Datei normal gestartet, brechen aber nach kurzer Zeit mit den programminternen Fehlermeldungen ab. Danach wird das nächste Programm

gestartet und bricht ebenfalls mit einer wüsten Fehlermeldung ab. Dieses Spiel treibt »SUBMIT«, bis die gesamte SUB-Datei abgearbeitet ist. Für den Anwender zeigt sich CP/M hier von seiner unfreundlichen Seite, doch keine Angst, die Daten auf den eingelegten Disketten nehmen durch solche Fehleingaben keinen Schaden.

Vorausgesetzt die SUB-Datei ist in Ordnung, erscheint am Bildschirm als erstes eine Auflistung aller »ASM«-Dateien. Danach wird automatisch der CP/M-Assembler »MAC« geladen, der die auf die Variable »$1« zugewiesene Datei assembliert. Die im »SUBMIT«-Aufruf angebene Datei muß dazu mit der Dateikennung »ASM« in Laufwerk A zu finden sein. »MAC« erstellt sodann die Dateien »DISK.HEX«, »DISK.PRN« und »DISK.SYM«. Von »HEXCOM«, das als nächstes geladen wird, wird ein ablauffähiges »COM«-File mit dem Namen »DISK.COM« erstellt. Die nächsten beiden Anweisungen der »SUB«-Datei benötigen eine formatierte Diskette in Laufwerk B. Dorthin werden die Dateien »DISK.PRN« und »DISK.SYM« kopiert. Diese neuen Dateien erhalten die Namen der zweiten Variablen »$2« und zwar »DRUCK.PRN« und »DRUCK.SYM«. Die nächste Anweisung in der kleinen »SUB«-Datei offenbart wieder eine kleine Besonderheit von »SUBMIT«: Nach dem Programmaufruf, in diesem Fall »PIP«, werden die benötigten Parameter in der »SUB«-Datei angegeben. Derartige Eingaben werden mit einer nach rechts geöffneten spitzen Klammer gekennzeichnet. Im obigen Beispiel werden die Druck- und Symboldatei auf Laufwerk B kopiert. Die letzte Klammer ohne Parameter kennzeichnet für PIP das Ende der Operation. Zum Schluß werden noch die Dateien »DISK.SYM« und »DISK.PRN« auf der Arbeitsdiskette in Laufwerk A gelöscht, da diese auf Laufwerk B gesichert wurden. Nach der Ausgabe des Disketteninhaltsverzeichnisses beendet »SUBMIT« seine Arbeit und überläßt die Kontrolle über das System wieder dem Anwender. Nun wissen Sie, wie man mit »SUBMIT« arbeitet. Doch das Rätsel um das im Handbuch so mager beschriebene »PROFILE.SUB« ist immer noch nicht gelüftet. Der Aufbau entspricht genau dem der bereits ausführlich besprochenen »SUB«-Datei. Doch nun zum eigentlichen Sinn der »PROFILE.SUB«. Beim »Booten« von CP/M sucht das Betriebssystem, bevor es die Kontrolle an den CCP (Kommandoprozessor) übergibt, als erstes die Datei »PROFILE.SUB« auf der Systemdiskette. Findet CP/M diese Datei, geschieht genau dasselbe, als wenn »PROFILE.SUB«, was auch ohne weiteres möglich ist, mit »SUBMIT« aufgerufen würde. Die dort angegebenen Programme werden geladen und ausgeführt. Für den Aufbau der Datei gelten die gleichen Bedingungen, wie für eine entsprechende durch »SUBMIT« aufzurufende Datei. Wo aber liegt der Nutzen für den Anwender? Die Vorteile liegen auf der Hand. Man kann beispielsweise beim Laden des Systems sofort das Datum setzen lassen, sowie die Tastaturkonfiguration ändern. Natürlich können alle zur Verfügung stehenden »COM«-Files aufgerufen werden. Eine weitere Möglichkeit wäre das Anfertigen von Sicherheitskopien unmittelbar nach dem Laden des Systems. Werden auf einer Diskette im Laufe einer Sitzung Dateien verändert, erweist sich dies oft als nützlich. Die Probleme, die das magere Handbuch gerade mit »SUBMIT« und der Datei PROFILE.Sub aufwirft, sind jetzt hoffentlich aus dem Weg geräumt, so daß einer effektiven Nutzung dieser komfortablen CP/M-Einrichtungen nichts mehr im Wege steht.

Die Sache mit den Usern

Wenn CP/M gebootet wird, befindet man sich automatisch in der User-Ebene Null (User=Benutzer, Anwender). Was hat es nun mit den anderen User-Bereichen, es gibt deren 16 (0-15), auf sich? Werden in einem anderen User-Bereich Daten gespeichert, legt CP/M ein weiteres Inhaltsverzeichnis für diese Ebene an. Die dort untergebrachten Daten können nur unter dieser User-Nummer bearbeitet werden. Wollen Sie beispielsweise unter Ebene drei arbeiten, so geben Sie einfach »USER« ein. Der Computer fragt dann nach der gewünschten Nummer.

Wie können nun diese User-Bereiche sinnvoll eingesetzt werden? Wenn sich nicht alles im Bereich Null abspielen soll, bietet sich hier die Möglichkeit an, etwas Organisation auf die Diskette zu bringen. So kann man beispielsweise im Bereich Null das System zum Booten speichern. Will man mit Wordstar und dBase auf einer Diskette arbeiten und dBase-Dateien in Wordstar laden, hat man die Möglichkeit, für dBase und Wordstar je einen eigenen Benutzerbereich anzulegen.

Der Trick mit »SYS«

Leider kann auf Dateien von anderen Benutzerebenen nicht ohne weiteres zugegriffen werden. Doch auch dafür gibt es eine Lösung. Eine solche gemeinsame Datei muß sich in der Bedienerebene Null befinden. Zusätzlich muß dieser Datei mit dem »SET«-Kommando ein »SYS«-Attribut zugeordnet werden, damit diese als bereichsübergreifende Systemdatei erkannt wird. Dazu ein Beispiel, in dem die oben erworbenen Kentnisse zu »SUBMIT« benötigt und vertieft werden. Als erstes muß natürlich die verwendete Diskette entsprechend vorbereitet werden. Das Ergebnis soll folgendermaßen aussehen: In der Benutzerebene Null liegt das System, sowie die dBase beiliegende Beispieldatei »BUECHER.DBF«. In den User-Bereich eins kopiert man alle nötigen Dateien, die zum korrekten Ablauf von dBase benötigt werden. Zu guter Letzt wird die Ebene zwei mit den nötigen Dateien zu Wordstar versorgt. Dann kann es praktisch schon losgehen. Doch anstatt jetzt umständlich zwischen den User-Ebenen hin und her zu schalten, werden wir für diesen Zweck eine, diesmal etwas größere, »SUBMIT«-Datei aufbauen (Bild 1). Diese »SUBMIT«-Datei soll nun folgenden Vorgang bearbeiten: Die Datei »BUECHER.DBF« wird als erstes in der Benutzerebene eins auf die von Wordstar zu lesende Datei »TEST.TXT« umgewandelt. Dazu muß in dBase nach dem Eröffnen der Datei mit »USE« folgende Anweisung erteilt werden:

```
COPY TO TEST DELIMITED WITH,
```

Damit wäre die Datei »TEST.TXT« erstellt. Danach muß dieses File in die Benutzerebene Null kopiert und mit dem Attribut »SYS« für Systemdatei versehen werden. Diese Aufgabe erledigt das »SET«-Kommando. Nun tritt in Bereich zwei Wordstar in Aktion. Zur Demonstration genügt es, wenn eine Zeile der eingelesenen Datei »TEST.TXT« verändert wird.

Nach dem Verlassen von Wordstar wird die Datei wieder in Bereich Null kopiert und mit dem »SYS«-Attribut versehen. Wo aber ist der praktische Nutzen dieses Beispiels zu sehen?

```
user 0
era test.txt
b:set buecher.dbf[rw,sys]
user 1
dbase
<$1
user 0
pip a:=a:test.txt[g1]
b:set test.txt[rw,sys]
user 1
era test.txt
user 2
ws
user 0
era test.txt
pip a:=a:test.txt[g2]
b:set test.txt[rw,sys]
user 2
era test.txt
user 0
```

Bild 1. SUB-Datei mit allen Schikanen

Directory For Drive B: User 0

Name		Bytes	Recs	Attributes		Prot	Update	Create
BUECHER	DBF	6k	44	Sys	RW	None		
CCP	COM	4k	25	Dir	RW	None		
CPM+	SYS	24k	184	Dir	RW	None		
D	TXT	0k	0	Dir	RW	None	08/05/86 03:13	08/05/86 12:58
PIP	COM	10k	68	Dir	RW	None		
SUBMIT	COM	6k	42	Dir	RW	None		
TEST	BAK	2k	3	Dir	RW	None	08/05/86 04:07	08/05/86 12:59
TEST	SUB	2k	3	Dir	RW	None	08/05/86 01:58	08/05/86 02:12
TEST	TXT	2k	11	Sys	RW	None	08/05/86 10:12	08/05/86 10:12

Total Bytes = 56k Total Records = 380 Files Found = 9
Total 1k Blocks = 52 Used/Max Dir Entries For Drive B: 25/ 128

Directory For Drive B: User 1

Name		Bytes	Recs	Attributes		Prot	Update	Create
DBASE	COM	20k	158	Dir	RW	None		
DBASEMSG	TXT	60k	475	Dir	RW	None		
DBASEOVR	COM	42k	328	Dir	RW	None		

Total Bytes = 122k Total Records = 961 Files Found = 3
Total 1k Blocks = 121 Used/Max Dir Entries For Drive B: 25/ 128

Directory For Drive B: User 2

Name		Bytes	Recs	Attributes		Prot	Update	Create
WS	COM	16k	128	Dir	RW	None		
WSMSGS	OVR	26k	194	Dir	RW	None		
WSOVLY1	OVR	34k	266	Dir	RW	None		

Total Bytes = 76k Total Records = 588 Files Found = 3
Total 1k Blocks = 75 Used/Max Dir Entries For Drive B: 25/ 128

Bild 2. Informatives Inhaltsverzeichnis mit DIR

Wenn Sie bestimmte Daten aus dBase-Dateien in Wordstar-Briefen verwenden wollen, können Sie so auf die dBase-Daten zugreifen. Wie das genau funktioniert, entnehmen Sie bitte der CP/M-Ecke aus der 64'er-Ausgabe 7/86. Doch nun wieder zum Ablauf der »SUBMIT«-Datei. Als erstes wird, falls nicht aktiv, User Null eingeschaltet. Die dort eventuell vorhandene Datei »TEST.TXT« wird gelöscht, um mögliche Schreib-Lesefehler zu vermeiden. Um von verschiedenen Benutzerbereichen auf »BUECHER.DBF« zugreifen zu können, muß diese Datei mit dem »SYS«-Attribut ausgestattet werden, was mit dem folgenden Set-Befehl erreicht wird. In Laufwerk B muß sich in diesem Fall die Systemdiskette befinden oder eine beliebige andere Diskette, die das Programm »SET« enthält. Nachdem dann in User Eins umgeschaltet ist, wird dort dBase geladen. Haben Sie beim Aufruf der SUB-Datei ein Datum mit angegeben, so wird dies automatisch durch $1 gesetzt, ansonsten wird die Datumseingabe übergangen. Sobald sich dBase mit dem Prompt (Punkt) meldet, führen Sie den oben angegebenen »COPY«-Befehl durch. In Ebene zwei haben Sie damit eine Datei Namens »TEST.TXT« erzeugt. Nachdem dBase mit »QUIT« verlassen wurde, kopiert »PIP« diese Datei in die Ebene Null, wo sie mit Hilfe von »SET« das SYS-Attribut erhält. Um Speicherplatz zu sparen, erfolgt als nächstes die Löschung der Datei »TEST.TXT« im Benutzerbereich eins. Nachdem in Ebene zwei geschaltet ist, wird sofort Wordstar geladen. Laden Sie zunächst »TEST.TXT« und verändern Sie einen beliebigen Satz. Nachdem Wordstar wieder verlassen ist, wird in Ebene Null die alte Testdatei gelöscht und die neue aus Ebene zwei herauskopiert. Nachdem sie dort wieder als Systemdatei gekennzeichnet ist, wird die Datei in Ebene zwei gelöscht und in den Bereich Null zurückgesprungen. Damit sollten jetzt alle noch bestehenden Probleme mit »SUBMIT«-Dateien und User-Bereichen gelöst und einer sinnvollen Anwendung dieser mächtigen CP/M-Komponenten nichts mehr im Wege stehen.

Timestamps, Initdir und Set

Diese drei sehr interessanten CP/M-Befehle bieten ein optimales Werkzeug für die interne Verwaltung der Diskettendateien. Die Möglichkeiten reichen von Datums- und Zeitprotokollierung der einzelnen Zugriffe bis hin zum Paßwortschutz für einzelne Dateien, Label und sogar des Systems. Wie komfortabel und hilfreich diese Optionen sein können, lernt man erst nach längerer Anwendung zu schätzen. So erhält man jederzeit Auskunft über die letzten Dateizugriffe und kann somit auch die große Palette der »DIR«-Optionen nützen. Doch jetzt zur näheren Erklärung und Beschreibung dieser Eigenschaften. Die Eintragung der sogenannten Timestamps, das sind die Zeit- und Datumseinträge, ist leider im Handbuch nicht sehr ausführlich beschrieben. Denn allein mit dem Kommando »INITDIR« ist es noch lange nicht getan. Zwar wird mit diesem Systemprogramm die Diskette für die Aufnahme von Timestamps vorbereitet, doch ganz ohne das Programm geht es auch wieder nicht. Also nicht vergessen, als erstes immer mit »INITDIR« die Diskette initialisieren. Die Timestamps können mit dem Kommando »DIR[DATE]« ausgegeben werden. Doch ein Versuch nach »INITDIR« führt lediglich zu einer Fehlermeldung, die besagt, daß die Diskette gar nicht für Timestamps vorbereitet ist. Was soll denn das? Die Diskette ist zwar für Einträge vorbereitet, aber die dafür vorgesehene Funktion des Systems ist noch nicht aktiviert. Dazu haben wir zum Glück ein Allround-Programm mit Namen »SET« zur Verfügung. Diese Funktionen werden speziell zum Setzen der Zeit- und Datumskennungsart benötigt. Drei Mög-

lichkeiten stehen zur Verfügung: Datum und Zeit der Entstehung soll festgehalten werden (Set[create=on]); Eintragen des letzten Dateizugriffs bei Dateien mit dem Read-Write-Attribut (Set[access=on]) und die Speicherung der Zeitdaten über die letzte Dateiveränderung (Set[update=on]). Doch Vorsicht, Create und Access schalten sich gegenseitig aus! Mit Update wird nur ein Zeiteintrag gemacht, wenn die Datei effektiv verändert wurde. Bei einem reinen Lesezugriff erfolgt bei Update keinerlei Aktion. Eine Diskette, die mit »INITDIR« initialisiert wurde und mit »SET[CREATE=ON, UPDATE=ON]« für Zeit- und Datumseinträge vorbereitet wird, gibt schließlich mit der entsprechenden »DIR«-Option ein optisch ansprechendes und vor allem informatives Bild ab (Bild 2). Ein Inhaltsverzeichnis dieser Art erhalten Sie, nachdem die Diskette für Zeit- und Datumseinträge aktiviert wurde und dann mit folgendem »DIR«-Befehl abgerufen wird:

```
DIR[DATE,USER=ALL,NOPAGE]
```

Sieht doch schon richtig professionell aus, oder etwa nicht? Der von CP/M erhobene Anspruch auf Professionalität muß sich schließlich irgendwo bestätigen.

»SET« kann noch einiges mehr. Eine der Optionen, die vor allem für Leute mit Top-Secret-Daten interessant ist, präsentiert sich mit einem vielfältigen Paßwortschutz in vielen Ebenen. Dafür sind allerdings ausnahmsweise genügend Informationen im Handbuch zu finden, so daß darauf hier nicht näher eingegangen werden muß.

Was nicht im Handbuch steht

Allerdings ist der Paßwortschutz nur dann 100prozentig, wenn das Directory-Label mit einem solchen versehen ist. In diesem Fall besteht keine Möglichkeit, den Schutz mit »SET [PROTECT=OFF]« auszuschalten, da immer wieder das lästige Paßwort abgefragt wird. Andernfalls können nach dem Abschalten des Schutzes alle Dateien ohne Probleme bearbeitet werden. Leider ist der Mensch nicht unfehlbar. Daher kommt es immer wieder vor, daß ein Paßwort vergessen wird. Handelt es sich dabei um den Schutz für das Disketten-Label, kommt anscheinend jede Hilfe zu spät. An diesem Punkt weichen wir total von der CP/M-Linie ab und wenden uns dem guten alten C64 zu. C64 und CP/M 3.0? Wer einen guten Diskettenmonitor für den C64 besitzt, ist aus dem Schneider. Denn, man will es beinahe nicht glauben, damit können auch CP/M-Disketten bearbeitet werden. Eine ideale Voraussetzung, um der eigenen Vergeßlichkeit ein Schnippchen zu schlagen. Allerdings ist eine CP/M-Diskette anders aufgebaut als die des C64. Sollte jemand auf die Idee kommen, die Directory in Block 18 zu suchen, wird er eine herbe Enttäuschung erleben. Das CP/M-Inhaltsverzeichnis findet sich immer in der den Systemblöcken unmittelbar folgenden Spur, also in Spur eins oder zwei. Dort liegt in den Sektoren null bis acht das eigentliche Directory. Bei der Editierung der entsprechenden Blöcke taucht auf dem Bildschirm ein wahrer Zeichenwirrwarr auf. Das sollte aber nicht weiter stören. Die Namenseinträge sind ohne weiteres zu finden. Soll nur das Paßwort einer bestimmten Datei gelöscht werden, fährt man in Sektor 15 oder 20 an den Anfangsbuchstaben und füllt mindestens 23 Byte mit dem Hexadezimalwert E5. Versuchen Sie nicht Ihr Paßwort zu finden, dieses befindet sich verschlüsselt hinter dem Dateinamen. Genauso wird das Paßwort für das Disketten-Label gelöscht. Dabei wird allerdings auch der Diskettenname überschrieben. Der kann dafür ohne Paßwort-Probleme neu angelegt werden. Sollten jetzt einige Hacker glauben, die Anschaffung von CP/M wäre eine lohnende Sache, schließlich kann endlich der Kopierschutz gebrochen werden, kann man nur einen der großen Vorteile der CP/M-Software erwidern: Ein Großteil der auf dem Markt erhältlichen Programme sind ohne jeglichen Kopierschutz erhältlich. Von der angebotenen Public-Domain-Software ganz zu schweigen.

Was tun mit dem Editor?

Ein vieldiskutiertes Thema ist und bleibt der zum System gelieferte Editor. Dieses Monstrum an angeblicher Eingabehilfe erweist sich bereits nach dem ersten Laden als völlig unkomfortabel. Dazu muß gesagt werden, daß dieser Editor aus den Anfangszeiten von CP/M stammt. Damals war das Fortschritt. Heute kann man ohne »ED« auskommen, wenn Software wie Wordstar und Turbo-Pascal zur Verfügung steht. Diese Editoren bieten nicht nur einen sehr hohen Komfort, sondern erzeugen ASCII-Dateien, die von jedem Compiler verarbeitet werden. Der Programmierer kann hier auf alle Vorteile der Textverarbeitung zurückgreifen, ohne sich sonderliche Nachteile einzuhandeln. Für diejenigen, die sich trotzdem mit dem CP/M-Editor zufriedengeben müssen, ein paar Hinweise und nähere Erläuterungen. Geladen wird der Editor mit angehängtem Dateinamen. Ist die angehängte Datei noch nicht auf Diskette, wird automatisch ein neues File angelegt. Ansonsten wird die gewünschte Datei geöffnet. Wohl gemerkt, sie wird nur geöffnet. Wenn direkt nach dem Aufruf einer der List-Befehle des ED angewandt wird, passiert nichts. Die Begründung liegt in der Natur des ED. Er ist ein sogenannter zeilenorientierter Editor. Das heißt, der Text kann nur zeilenweise bearbeitet werden. Full-Screen-Editoren dagegen lassen ein Bewegen des Cursors an eine beliebige Stelle des Bildschirms zu. Weiterhin arbeitet der CP/M-Editor mit Datenpuffern. Dieser Datenpuffer ist lediglich ein Speicherbereich, in dem der Text zwischengespeichert wird. Es wird also nicht auf der Diskette gearbeitet, sondern direkt im freien Speicherplatz. Nach dem Starten des ED mit Dateinamen ist dieser Speicher prinzipiell leer. Mit dem Editorbefehl #A wird der Puffer von der Diskette mit Daten versorgt. Erst dann kann der Anwender den angeforderten Text bearbeiten. Doch Vorsicht! Verwenden Sie nicht sofort die Einfügeanweisung »I«. Der ED-interne Zeilenzeiger steht nach dem Laden des Textes immer auf Zeile eins. Um ans Text- beziehungsweise Pufferende zu springen, muß der Cursor erst mit der Anweisung »-B« auf das Ende gestellt werden. Danach kann dann mit dem Einfügen begonnen werden. Ein weiteres Problem stellt die Positionierung des Cursors innerhalb einer Zeile dar. Eine eingegebene Zeile kann leider nicht ohne weiteres editiert werden. Sie wollen beispielsweise folgende Zeile ändern: »Sehr gehrte Damen und Herren,«. Dazu muß erst auf der entsprechenden Stelle positioniert werden, um ein »e« einzugeben. Positioniert wird auf den Buchstaben, nach dem etwas eingefügt wird. In diesem Fall lautet der Befehl »7C«. Jetzt kann mit »I« der fehlende Buchstabe erfaßt werden. Die editierte Zeile muß mit »CONTROL-Z« verlassen werden. Erfolgt der Abbruch mit »RETURN«

Byte	Beschreibung
0	Laufwerksnummer
1 - 8	Dateiname
9 - 11	Extension
12	Nummer des Eintrags
13 - 14	Intern reserviert
15	abgelegte Records
16 - 31	Nummern der für die Datei belegten Blöcke
32	nächster Record
33 - 34	absoluter Record bei Direktzugriff
35	Überlauf-Flag für BDOS-Funktion 35

Bild 3. Der FCB-Block von CP/M



hängt der ED ein Zeilenendezeichen an. ED schneidet den hinteren Teil der Zeile ab und bildet mit dem Rest eine neue Zeile. Zum Löschen existieren zwei Befehle. Die Anweisung »K« löscht innerhalb einer Zeile, während mit »D« ganze Zeilen gelöscht werden. Wenn es bei diesen beiden Anweisungen zu Verwechslungen kommt, wird der Text stark verändert. Man wirft besser doch noch einen Blick ins Handbuch, um solche nervenaufreibenden Fehler zu vermeiden.

Die Innereien

Vor allem jene, die gerne mehr über ein System wissen oder es gar manipulieren wollen, finden im folgenden nützliche Hinweise.

Ein Directory-Eintrag unter CP/M kann höchstens 16 KByte Diskettendaten verwalten. Für alle weiteren angefangenen 16 KByte ist jeweils ein weiterer Directory-Eintrag vonnöten. Sicher haben Sie schon einmal bemerkt, daß es nach einem Diskettenwechsel länger als normal dauert, die Directory auszulesen. Dafür gibt es eine systembedingte Erklärung. Im Arbeitsspeicher des Computers befindet sich der sogenannte File-Control-Block (FCB), in den ein Directory-Eintrag aufgenommen wird. Im FCB (Bild 3) sind beispielsweise die Laufwerksnummer, der Dateiname, die Anzahl der abgelegten Records oder die Nummer des aktuellen Records gespeichert. Er repräsentiert praktisch den momentan aktuellen Zustand einer in Bearbeitung befindlichen Datei.

Ohne dieses Modul läuft unter CP/M absolut nichts. Das BDOS ist der fleißigste Arbeiter im System. Vor allem für den Assemblerprogrammierer bietet das BDOS Möglichkeiten, die eine effektive Programmierung mit minimalem Aufwand erlauben. Aufgebaut ist das BDOS aus einer Menge von einzelnen Funktionen, die Ein-/Ausgabe zwischen den einzelnen Geräten steuern. Jede Funktion hat ihre eigene Nummer, unter der sie im Maschinenprogramm aufgerufen werden kann. Die Einsprungadresse ins BDOS ist $0005 hexadezimal. Die Nummer der gewünschten Funktion wird in einem Register übergeben, ebenso die Adresse der zu bearbeitenden Speicherstelle. Der Assemblerprogrammierer muß sich um keine einzige Ein-/Ausgaberoutine kümmern. Er hat nur dafür zu sorgen, daß die richtigen Parameter der gewählten Funktion übergeben werden. Im Anschluß folgt eine Übersicht der BDOS-Funktionen.

Die BDOS-Funktionen

Als erstes wird die Funktionsnummer genannt, danach die Bedeutung und die benötigten Parameter.

0 System Reset
Warmstart; Rücksprung zum System.
Parameter: 00H in Register: C

1 Console Input
liest ein Zeichen von der Konsole.
Parameter: 01H in Register: C
Gedrücktes Zeichen: in Register: A

2 Console Output
Gibt ein Zeichen auf den Bildschirm aus.
Parameter: 02H in Register: C
Auszugebendes Zeichen: in Register: E

3 Auxiliary Input
Liest ein Zeichen aus dem AUXIN-Kanal.
Parameter: 03H in Register: C
Gelesenes Zeichen: in Register: A

4 Auxiliary Output
Legt ein Zeichen auf den AUXOUT-Kanal.
Parameter: 04H in Register: C
Auszugebendes Zeichen: in Register: E

5 List Output
Gibt ein Zeichen auf den Drucker aus.
Parameter: 05H in Register: C
Auszugebendes Zeichen: in Register: E

6 sollte nicht verwendet werden

7 Auxiliary Input Status
Zeichen zur Eingabe an AUXIN?
Parameter: 07H in Register: C
Ergibt in Register A: FFH für ja
00H für nein

8 Set lobyte
Zeichen zur Ausgabe an AUXOUT?
Parameter: 08H in Register: C
Ergibt in Register A: FFH für ja
00H für nein

9 Print String
String, mit $ abgeschlossen, auf den Bildschirm ausgeben.
Parameter: 09H in Register: C
Stringadresse: in Register: DE

10 Read Console Buffer
Liest einen String von Konsole ein.
Parameter: 0AH in Register: C
Pufferadresse: in Register: DE

11 Get Console Status
Prüft, ob Zeichen von Tastatur eingegeben wurden.
Parameter: 0BH in Register: C
Ergibt in Register A: 00H für nein
01H für ja

12 Return Version Number
Gibt die Nummer der verwendeten Version zurück.
Parameter: 0CH in Register: C
Rückgabe: in Register: HL
H = 0 = CP/M 80
Nummer in L: BDOS-Versionsnummer

13 Reset Disk System
Die Datenpufferadresse (DMA) bei Laufwerk A wird auf 80H gesetzt.
Parameter: 0DH in Register: C

14 Select Disk
setzt aktuelles Laufwerk.
Parameter: 0EH in Register: C
Laufwerk (A=1,B=2,...) in Register: E
Gibt in Register A ein Error-Flag und in Register H einen physikalischen Fehler zurück.

15 Open File
Öffnet eine Datei.
Parameter: 0FH in Register: C
FCB-Adresse: in Register: DE
Directory-Code: in Register: A
Fehler: in Register: H

16 Close File
Schließt eine Datei.
Parameter: 10H in Register: C
Ansonsten wie Funktion 15.

17 Search For First
Sucht den ersten Eintrag einer Datei im Directory.
Parameter: 11H in Register: C
FCB-Adresse in Register: DE

18 Search For Next
Sucht nach Ausführung von Funktion 17 den nächsten Eintrag.
Parameter: 12H in Register: C

19 Delete File
Löscht nicht schreibgeschützte Dateien.
Parameter: 13H in Register: C
FCB-Adresse in Register: DE
Gibt in H einen Diskettenfehler und in A den Directory-Code zurück.

20 Read Sequential
Liest den folgenden Record einer Datei.
Ergebnis kommt in den Datenpuffer.
Parameter: 14H in Register: C
FCB-Adresse in Register: DE
Gibt in A einen Fehler-Code und in H einen physikalischen Fehler zurück.

21 Write Sequential
Beschreibt den folgenden Record aus dem Datenpuffer.
Parameter: 15H in Register: C
FCB-Adresse in Register: DE
Fehlerrückgabe wie Funktion 20.

22 Make File
Legt ein neues File an. Da noch keine Daten vorhanden sind, kommt in die Directory ein Leerstring.
Parameter: 16H in Register: C
FCB-Adresse in Register: DE

23 Rename File
Umbenennen von Dateien. Im FCB müssen die ersten 16 Byte den alten und die restlichen 16 Byte den neuen Namen enthalten.
Parameter: 17H in Register: C
FCB-Adresse in Register: DE

24 Return Login Vector
Fragt die angeschlossenen Laufwerke ab. Gibt einen 16-Bit-Wert in HL zurück, wobei das niederwertige Bit für Laufwerk A steht und das höchstwertige für P.
Parameter: 18H in Register: C
Rückgabe Login Vector in Register: HL

25 Return Current Disk
Gibt das aktuelle Laufwerk an. Laufwerk A=0, B=1, usw.
Parameter: 19H in Register: C
Rückgabe Laufwerk in Register: A

26 Set DMA Address
Datenpufferadresse für Schreib-/Lesezugriffe setzen.
Parameter: 1AH in Register: C
DMA-Adresse in Register: DE

27 Get Addr (Alloc)
Adresse des Blockbelegungs-Verzeichnisses holen. Wird von SHOW zur Ermittlung des freien Speicherplatzes verwendet.
Parameter: 1BH in Register: C
Adresse zurück: in Register: HL

28 Write Protect Disk
Das gewählte Laufwerk erhält einen temporären Schreibschutz.
Parameter: 1CH in Register: C

29 Get Read-Only Vector
Ermittelt schreibgeschützte Laufwerke.
Parameter: 1DH in Register: C

30 Set File Attributes
Aktiviert Dateiattribute, oder setzt diese zurück. Überträgt entsprechenden FCB-Eintrag.
Parameter: 1EH in Register: C
FCB-Adresse in Register: DE

31 Get DPB-Address
Sucht die Adresse des Disk-Parameter-Blocks.
Parameter: 1FH in Register: C
Die Adresse wird ins HL-Register gebracht.

32 Set/Get User Code
Setzen oder Abfragen der aktuellen Benutzernummer.
Parameter: 20H in Register: C
0FFH in Register: E
In Register A wird die aktuelle Benutzernummer zurückgegeben.

33 Read Random
Lesen eines Records im Direktzugriff.
Parameter: 21H in Register: C
FCB-Adresse in Register: DE
In den Registern A und H werden die Fehlermeldungen zurückgegeben.

34 Write Random
Schreiben eines Records im Direktzugriff.
Parameter: 22H in Register: C
sonst wie Funktion 33.

35 Compute File Size
Dateigröße in Records ermitteln. Die Größe wird in den Erweiterungsbytes für relativen Dateizugriff des FCB abgelegt.
Parameter: 23H in Register: C
FCB-Adresse in Register: DE
Fehler werden in Register A und H zurückgegeben.

36 Set Random Record
Momentane Record-Nummer ermitteln und im FCB ablegen.
Parameter: 24H in Register: C
FCB-Adresse in Register: DE

37 Reset Drive
Reset für einzelnes Laufwerk durch Übergabe eines 16-Bit-Wertes. Niederwertiges Bit ist Laufwerk A, höchstwertiges ist P.
Parameter: 25H in Register: C
Laufwerksvektor in Register: DE

38 Nur für MP/M
39 Nur für MP/M

40 Write Random With Zero
Wie Funktion 34, jedoch mit vorhergehendem Auffüllen des Records mit Nullen.
Parameter: 28H in Register: C
FCB-Adresse in Register: DE
In A und H werden Fehler zurückgegeben.

41 Nur für MP/M
42 Nur für MP/M
43 Nur für MP/M

44 Set Multi-Sector Count
Während ständiger Schreib-/Lesevorgänge des BDOS können maximal 128 Records auf einmal bearbeitet werden.
Parameter: 2CH in Register: C
Anzahl der Sektoren in Register: E

45 Set BDOS Error Mode
Physikalische und erweiterte Fehler werden abgefangen.
Register E=255: Return Error Mode
Register E=254: Return and Display Mode
Parameter: 2DH in Register: C
BDOS-Fehler-Modus in Register: E

46 Get Disk Free Space
Auf Disketten mit RW-Attribut wird der freie Speicherplatz ermittelt.
Parameter: 2EH in Register: C
Laufwerksnummer in Register: E
Die Größe des freien Speichers in Records steht in den ersten drei Byte der DMA.

47 Chain to Program
Overlay von Programmen. Die Befehlszeile muß sich im DMA-Puffer befinden und mit 00H abgeschlossen sein. Register E muß auf FFH gesetzt werden.
Parameter: 2FH in Register: C
Overlay-Flag in Register: E

48 Flush Buffers
Die Pufferinhalte werden recordweise auf die Diskette abgespeichert.
Parameter: 31H in Register: C
Flag f. Puffer löschen Register: E
Zum Löschen Register E auf FFH setzen.

49 Get/Set System Control Block (SCB)
Lesen und Schreiben des SCB.
Parameter: 31H in Register: C

50 Direct Bios Calls
Aufruf von BIOS-Routinen, die in Bank 0 abgelegt sind.
Parameter: 32H in Register: C
BIOS-Funktions-Nr. in Register: DE

59 Load Overlay
Laden von RSX-Modulen.
Parameter: 3BH in Register: C
FCB-Adresse in Register: DE
In den Registern A und H werden Fehler zurückgegeben.

60 Call Resident System Extension
Aufruf einer RSX-Systemerweiterung.
Parameter: 3CH in Register: C
RSX-Funktions-Nr. in Register: DE

98 Free Blocks
Alle Blöcke, die während den BDOS-Aktivitäten vorübergehend angelegt wurden, werden gelöscht. Vorsicht: Alle Dateien mit Close schließen.
Parameter: 62H in Register: C
Fehler werden in den Registern A und H abgelegt.

99 Truncate File
Dem letzten Record einer Datei wird eine wahlfrei Nummer im FCB eingetragen.
Parameter: 63H in Register: C
FCB-Adresse in Register: DE

100 Set Directory Label
Setzen oder Ändern des Directory-Labels.
Parameter: 64H in Register: C
FCB-Adresse in Register: DE

101 Return Directory Label Data
Im Label wird nach dem Byte gesucht, das Informationen über Paßwortschutz und Timestamps enthält. Register A wird mit diesem Register geladen.
Parameter: 65H in Register: C
Laufwerk: in Register: E

102 Read File Date Stamps and Paßwort Mode
Der Paßwort-Modus und Timestamps für Dateien wird ermittelt. FCB muß vorher mit dem Dateinamen belegt werden.
Parameter: 66H in Register: C
FCB-Adresse in Register: DE

103 Write File XFCB
Schreiben oder aktualisieren eines speziell für geschützte Dateien erstellten Directory-Eintrags.
Parameter: 67H in Register: C
FCB-Adresse in Register: DE

104 Set Date and Time
Stellen der internen Uhr. Übergabe der Zeit in Register DE im BCD-Format.
Parameter: 68H in Register: C
Dat-Adresse in Register: DE

105 Get Date and Time
Interne Uhr abfragen.
Parameter: 69H in Register: C

Die Zeit wird im BCD-Format in Register DE übergeben.

106 Set Default Password
Vorgabe eines Paßworts, bevor auf eine geschützte Datei zugegriffen wird.
Parameter: 6AH in Register: C
Paßwort-Adresse in Register: DE

107 Return Serial Number
Feststellen der CP/M-Seriennummer.
Parameter: 6BH in Register: C
Die Nummer wird in einem 6-Byte-Feld gespeichert, dessen Adresse in Register DE steht.

108 Get/Set Program Return Code
Return-Code für Programme ermitteln oder setzen. Neuen Code in Registerpaar DE übergeben, um den alten Wert auszulesen, DE auf FFFFH setzen.
Parameter: 6CH in Register: C

109 Get/Set Console Mode
Konsolen-Modus ermitteln oder neu setzen. Dazu muß ein 16-Bit-Wert in DE stehen. Bei Abfrage muß DE gleich FFFFH sein. Die Informationen finden sich in HL.
Parameter: 6DH in Register: C

110 Get/Set Output Delimiter
Die Ausgabe-Begrenzung für Strings wird gesetzt oder ermittelt (Standard=$). Bei Abfrage Register DE auf FFFFH setzen. Das Ergebnis befindet sich in Register A.
Parameter: 6EH in Register: C
Zum Setzen: Wert in Register: E

111 Print Block
Ein im Console-Control-Block definierter String, dessen Adresse in DE liegt, wird nach CONOUT gesendet.
Parameter: 6FH in Register: C
CCB-Adresse in Register: DE

112 List Block
Hat dieselbe Wirkung wie Funktion 111.
Es wird lediglich auf LST: ausgegeben.

152 Parse Filename
Wandelt einen von der Konsole eingegebenen Dateinamen in das interne FCB-Format um. DE enthält die Adresse eines Puffers mit folgenden Daten:
0 und 1 = Adresse des vorgegebenen Strings
2 und 3 = Adresse des FCB

Übersicht BDOS-Funktionen (Schluß)

Diese Auflistung, die alle Funktionen des BDOS enthält, ist nicht nur für den Assemblerprogrammierer von Nutzen. Dem Anwender hilft sie vor allem beim Analysieren von Systemfehlern. Bei den CP/M-Fehlermeldungen wird immer nur die Funktionsnummer ausgegeben, in der der Fehler auftrat. Anhand dieser Nummer kann die eigentliche Ursache für den Programmabbruch festgestellt werden. Hinter der angegebenen Funktionsnummer verbirgt sich der Schlüssel zu obiger Auflistung. Dort brauchen Sie nur noch unter der entsprechenden Funktionsnummer nachzusehen und schon haben Sie die genaue Ursache für den Ausstieg. Wenn sich das System beispielsweise mit einem Fehler in Funktion 15 meldet, sieht man auf einen Blick, daß eine Datei nicht eröffnet werden konnte. Der Programmierer fragt sich zu Recht, wie diese Routinen in eigenen Programmen verwendet werden. Die angegebenen Register werden mit den erforderlichen Daten gefüllt. Die Funktionsnummer wird dabei immer im Register C übergeben. Die sonst erforderlichen Daten werden in einem weiteren Register oder Registerpaar gespeichert. Danach wird mit »CALL 5« das BDOS aufgerufen. So können alle Routinen des BDOS im eigenen Programm verwendet werden. Man spart sich die Zeit zum Schreiben eigener entsprechender Routinen, da man durch die Vielfalt der BDOS-Routinen bereits eine große Makrobibliothek zur Verfügung hat. Noch ein Tip für Assemblerprogrammierer, die Texte als Konstanten verwenden. In den ersten beiden Byte des definierten Puffers werden Informationen zur Stringlänge abgelegt. Im ersten Byte findet sich die Definitionslänge, im zweiten die wirkliche Länge des Strings. Bei Ausgaben über die BDOS-Funktion 9 ist zu beachten, daß am Ende des Strings ein Dollar-Zeichen steht, da dieses als Endemarke verwendet wird. Sollen von Konsole eingelesene Strings (Funktion 10) wieder ausgegeben werden, müssen diese erst mit dem Dollarzeichen am Ende versehen werden. Die Informationen über die Länge der Konstanten befinden sich, wie bereits erwähnt, in den ersten beiden Bytes.

Das BIOS – Schichtarbeit im Untergrund

Auf einer weit niedrigeren Ebene als BDOS arbeitet eine weitere CP/M-System-Komponente: das BIOS. Es ist für die Steuerung der Peripherie und der computerinternen Hardware zuständig. Der Zugriff auf die BIOS-Routinen erweist sich als weitaus schwieriger als beim BDOS. Zum Glück können die meisten benötigten Operationen mit den BDOS-Funktionen ausgeführt werden. Schwierig wird es allerdings, wennn man zum Beispiel auf einzelne Blöcke und Sektoren der Diskette zugreifen will. Dies kann nur unter Verwendung der BIOS-Routinen gelöst werden. Nachfolgend eine Liste der Funktionen:

Nr.	Funktion
0	Kaltstart
1	Warmstart
2	Konsolenabfrage (A=0: keine Taste, sonst 255)
3	Zeichen von Konsole nach A
4	Zeichen von C an Konsole
5	Zeichen von C an Drucker
6	Zeichen von C an Hilfskanal
7	Zeichen von Hilfskanal an A
8	Schreib-/Lesekopf auf Spur 0 stellen
9	Laufwerk anwählen (DPH-Adresse in HL)
10	Spur anwählen (Nummer in BC)
11	Sektor anwählen (Nummer in BC)
12	Adresse (in BC) für DMA setzen
13	Sektor nach anwählen lesen
14	Sektor nach anwählen schreiben
15	Druckerstatus abfragen
16	Sektornummer (in BC) anhand von Skewing-Tabelle (Beginn in DE) übersetzen (Ergebnis in HL)
17	Konsolen-Status prüfen
18	Eingabe-Status für Hilfskanal prüfen
19	Ausgabe-Status für Hilfskanal prüfen
20	Adresse für I/O-Gerätenamen-Tabelle ermitteln
21	Physikalische Geräte in Tabelle initialisieren
22	DPH-Adresse für Laufwerk ermitteln (nach HL)
23	Fortlaufende Sektoren für Read and Write zählen
24	Löscht physikalischen Sektorpuffer
25	Blockverschiebung: von Adresse in DE nach Adresse in HL, Anzahl der Byte in Registerpaar DC
26	Zeit setzen/abfragen
27	Speicherbank (Nummer in A) anwählen
28	Speicherbank (Nummer in A) für DMA-Operationen anwählen
29	Zielbank (Reg. B) und Quellbank (Reg. C) für Funktion 25, bei Verschiebung zwischen verschiedenen Banken

Übersicht BIOS-Funktionen

DPH bedeutet Disk-Parameter-Header, eine Tabelle, die wichtige Informationen über die Steuerung der Laufwerke enthält. DPB steht für Disk-Parameter-Block, dieser stellt die nötigen Disketteninformationen zur Verfügung. Beide Tabellen liegen in Bank 0, so daß nicht unmittelbar darauf zugegriffen werden kann.

Um diese BIOS-Funktionen für eigene Programme zu nutzen, verwendet man die BDOS-Funktion 50. Allerdings muß

vorher eine Tabelle (Bild 4) im Speicher kreiert werden, deren Adresse dem BDOS in Register DE übermittelt werden muß. Gibt das BIOS Werte zurück, stehen diese nach Ausführung in den entsprechenden Registern. Interessant erscheint die Möglichkeit, direkt auf die einzelnen Sektoren auf Disk zuzugreifen. Dazu bietet BDOS keine Funktion an, so daß auf BIOS auszuweichen ist.

Direkte Systemmanipulation

Im BDOS existiert eine Tabelle, in der wichtige Informationen für alle Systemteile abgelegt sind. Dieses nur 100 Byte lange Schlüsselelement nennt sich System-Control-Block (Bild 5). Zwar verändern viele BDOS-Aufrufe diese Tabelle, der Anwender kann aber über die BDOS-Funktion 49 darauf zugreifen. Dabei sollte mit äußerster Vorsicht vorgegangen werden, da unsachgemäße Einträge zum Systemabsturz führen. Dazu wird wieder eine Tabelle zur Verfügung gestellt, die in der TPA gespeichert ist. Byte null enthält das angesprochene SCB-Element. Byte eins enthält die Anweisung für die BDOS-Funktion. Soll ein Byte des SCB gesetzt werden, erhält Byte eins den Wert FFH (»H« steht immer für hexadezimal), soll ein Wort gesetzt werden, den Wert FEH, soll ein Wert abgefragt werden, erhält Byte eins den Wert 00H. Das Ergebnis der Abfrage wird in Byte zwei bei 1-Byte-Werten, und in Byte zwei und drei bei 2-Byte-Werten abgelegt.

Byte	Beschreibung
0	BIOS-Funktionsnummer
1	Inhalt Register A
2 und 3	Inhalt Registerpaar BC
4 und 5	Inhalt Registerpaar DE
6 und 7	Inhalt Registerpaar HL

Bild 4. Übergabetabelle für BDOS-Funktion 50

Alle, die mit dem Z80- oder 8080/8085-Assembler umzugehen verstehen, haben jetzt die Informationen erhalten, um sogar größere Systemmanipulationen vornehmen zu können. Der CP/M-Einsteiger kann sein System effektiver und mit weniger Fehlern als bisher nutzen. Vor allem ist bewiesen, daß CP/M zwar kein Betriebssystem zum Spielen ist, sich aber hervorragend für ernsthafte Anwendungen eignet. In der 64'er werden wir in Zukunft des öfteren einige CP/M-Programme, von denen es Tausende gibt, für Sie testen und vorstellen. (rf)

Nummer	Beschreibung
00H - 04H	Reserviert
05H	BDOS-Versions-Nummer
06H - 09H	User Flags
0AH - 0FH	Reserviert
10H - 11H	Program Error Return Code
12H - 19H	Reserviert
1AH	Anzahl der Bildschirmspalten
1BH	Aktuelle Bildschirmzeile
1CH	Seitenlänge Bildschirm
1DH - 21H	Reserviert
22H - 23H	CONIN-Flag
24H - 25H	CONOUT-Flag
26H - 27H	AUXIN-Flag
28H - 29H	AUXOUT-Flag
2AH - 2BH	LSTOUT-Flag
2CH	Page-Modus
2DH	Reserviert
2EH	Flag für CTRL-H = DEL
2FH	Flag für DEL = CTRL-H
30H - 32H	Reserviert
33H - 34H	Konsolen-Modus (BDOS 109)
35H - 36H	Reserviert
37H	Ausgabe-Delimiter (BDOS 110)
38H	Flag für Druckerausgabe
39H - 3BH	Reserviert
3CH - 3DH	Derzeitige DMA-Adresse
3EH	Aktuelles Laufwerk
3FH - 43H	Reserviert
44H	Aktuelle Benutzernummer
45H - 49H	Reserviert
4AH	BDOS-Multi-Sector-Count (BDOS 44)
4BH	BDOS-Error-Mode (BDOS 45)
4CH - 4FH	Suchpfad für 4 Laufwerke (SETDEF)
50H	Laufwerk für temporäre Dateien
51H	Laufwerk, das letzten Fehler meldete
52H - 56H	Reserviert
57H	Flag für Ausgabeart der BDOS-Meldungen
58H - 59H	Tage seit dem 1. Januar 1978
5AH	Stunden im BCD-Format
5BH	Minuten im BCD-Format
5CH	Sekunden im BCD-Format
5DH - 5EH	Basisadresse des Common-Speicherbereichs

Bild 5. Die SCB-Block-Beschreibung

ROCKUS







Druckeranpassung für Wordstar 3.0, dBase II und Multiplan

Am Beispiel von Wordstar 3.0 und dem Drucker NL-10 von Star wird hier gezeigt, wie einfach man die Möglichkeiten eines guten Textverarbeitungssystems nutzen und auf die individuellen Bedürfnisse anpassen kann.

Durch die folgende Druckeranpassung an Wordstar 3.0 sind Sie in der Lage, sowohl die speziellen Fähigkeiten Ihres Druckers auszunutzen, als auch die gewonnenen Kenntnisse auf die Programme dBase II und Multiplan C 128 zu übertragen, um so in allen drei Programmsystemen Ihren Drucker optimal verwenden zu können.

Druckeranpassung für Wordstar 3.0

Das Bedienungsfeld des NL-10 (control panel) bietet fantastische Möglichkeiten der Druckbildgestaltung. Wenn man aber schon über ein so ausgereiftes Textverarbeitungsprogramm wie Wordstar 3.0 verfügt, möchte man natürlich auch alle Möglichkeiten des Druckers durch Steuerbefehle nutzen können. Wer möchte nicht gerne durch sein Textverarbeitungsprogramm den Drucker veranlassen, zum Beispiel Schönschrift, Proportionalschrift, Breitschrift, Engschrift, Kursivschrift, Doppeldruck, Fettdruck oder gar zwei- oder vierfache Schrifthöhe wie von Geisterhand zu wechseln?

Ein Kopfbogenentwurf wie in Bild 1 dürfte doch sicher auch den anspruchsvollen Benutzer begeistern.

Wordstar bietet zwar einiges; die Möglichkeiten, zusätzliche Steuerzeichen an den Drucker zu geben, sind aber doch begrenzt. Mit gewissen Tricks und Änderungen über das Installationsprogramm läßt sich aber viel erreichen. In einer einfach gehaltenen Anleitung sollen diese Änderungen und Anpassungsarbeiten nachfolgend beschrieben werden. Sie beziehen sich auf den parallel am Userport angeschlossenen NL-10, da Wordstar 3.0 nur auf Druckern mit einer Centronics-Schnittstelle optimal anzupassen ist. Eine DIN-Belegung einer solchen Verbindung zwischen C 128 und dem Drucker können Sie Bild 2 entnehmen.

Anpassung

Die einzelnen Steuerzeichen für den Drucker sind auf der Wordstar-3.0-Systemdiskette in der Datei »WS.COM« vorhanden. Zusätzliche oder neue Steuerzeichen können in diese Datei eingebaut werden. Hierzu benötigen wir das ebenfalls auf der Wordstar-3.0-Diskette vorhandene Installationsprogramm »INSTALL.COM« und die dazugehörigen Daten »WS.INS«.

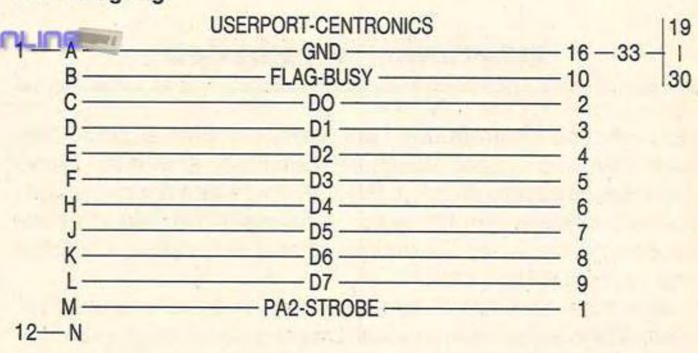


Bild 2. PIN-Belegung für ein Userport/Centronics-Kabel

COMPUTERSERVICE GmbH & Co KG
Handelsgesellschaft Hamburg
Alleiniger Geschäftsführer Dir. Alexander Interface

Bankverbindung Haspa 200 500 60/Nr. 4050
Tel. 040/333333
Postfach 301080
Druckerstr. 5, 2000 Hamburg 1
Hamburg , den

Computerservice Hamburg, Postfach 301080, 2000 Hamburg 1

Bild 1. Entwurf eines Kopfbogens mit Wordstar 3.0 sowie dem Drucker NL-10

Um andere Dateien nicht zu schädigen, formatieren wir eine neue Diskette und kopieren unter CP/M 3.0 mit dem Kopierprogramm »PIP.COM« von den Wordstar-Systemdisketten die drei Dateien:

INSTALL.COM	(Diskette 2)
WS.COM	(Diskette 1)
WS.INS	(Diskette 2)

Dies erreichen Sie bei einem Laufwerk durch »e:=a: Programmname. Kennung«, bei zwei Laufwerken durch »Ziellaufwerk := Quellaufwerk: Programmname. Kennung«. Legen Sie ins Ziellaufwerk die leere (aber formatierte) Diskette und in das Quellaufwerk die Wordstar-3.0-Diskette ein. Wenn Sie alle Risiken beim Kopieren ausschalten möchten, müssen Sie dem zuvor genannten noch »[ov]« anhängen. Die eckigen Klammern schließen die sogenannte Option ein, »o« bewirkt in jedem Fall ein Kopieren bis zum Ende der Datei (File - Ende - Markierungen werden überlesen), »v« führt einen Verify durch, das heißt die Daten werden nochmals überprüft.

Mit der so erstellten Diskette wollen wir jetzt die Neuinstallation durchführen. Wir starten mit dem Befehl: INSTALL <RETURN>.

Es erscheint jetzt auf dem Bildschirm eine Titel- und Copyright-Meldung, die mit dem Satz »Type any key to continue...« endet. Ein beliebiger Tastendruck führt uns also weiter. Die nächste Bildschirmmeldung zeigt uns, wozu INSTALL benötigt wird. Diese Anzeige verlassen wir wieder mit <RETURN>. Nun werden wir gefragt, welches Programm wir ändern wollen. Wir geben »WS« für Wordstar ein und gehen mit <RETURN> weiter. In der nächsten Bildschirmanzeige werden wir einige Benutzerhinweise finden, die wir aber im Augenblick nicht brauchen, also gehen wir mit einem beliebigen Tastendruck weiter.

Installationsvorarbeiten

Nun wird die Eingabe des Laufwerkes, in dem sich die Diskette mit der Datei WS.COM befindet, erwartet. Diese Anzeige verlassen wir mit <RETURN>, wenn nur das eingebaute Laufwerk benutzt wird. Verwenden Sie jedoch zwei Laufwerke, so geben Sie entsprechend »a:« oder »b:«, gefolgt von <RETURN>, ein.

Nun stellt sich die Frage, wie das zu installierende Programm bezeichnet werden soll. Da wir die bereits grundinstallierte Datei WS.COM nur ändern wollen, geben wir hier ein: WS.COM <RETURN>. Sie brauchen sich an dieser Stelle keine Sorgen zu machen, daß Ihre Installationsdatei überschrieben wird, da die Änderungen nur auf der Kopie der Originaldiskette durchgeführt werden. Es besteht also jederzeit die Möglichkeit, auf die Originaldatei zuzugreifen.

Wir erhalten jetzt die Meldung, daß die Datei WS.COM zur Grundlage genommen wird. Gleichzeitig wird vorgeschlagen, die neue Datei wieder WS.COM zu nennen. Diesem Vorschlag folgen wir durch Eingabe von <RETURN>.

Als nächstes werden wir durch die Bildschirmanzeige darauf hingewiesen, daß WS.COM auf der im Laufwerk befindlichen Diskette bereits existiert und - wenn wir fortfahren - überschrieben wird. Da wir ja eine neue WS.COM-Datei erzeugen wollen, sind wir einverstanden und drücken <RETURN>. Die folgende Anzeige ist eine Sicherheitsabfrage, die wir wieder mit <RETURN> bestätigen. Nach dem Hinweis, daß WS.COM kopiert wird, erscheint das Hauptmenü mit den Menüpunkten A - E und X. Da wir eine eigene Druckeranpassung vornehmen wollen, wählen wir <D> (Custom Installation of Printers). Wir erhalten nun einige Erläuterungen zum Druckerinstallationsmenü und verlassen es mit einer beliebigen Taste. Jetzt erscheint auf unserem Bildschirm das Druckerinstallationsmenü mit den Unterpunkten A - Q und X, das man, gemäß Bild 3 übersetzen kann.

Für die Verbesserung der Anpassung von Wordstar 3.0 an den Star NL-10 benötigen wir nur einige Menüpunkte, da die meisten keiner Veränderung bedürfen. Normalerweise können nur benutzereigene Funktionen unter dem Punkt <O> eingegeben werden. Hier gibt es vier Möglichkeiten: Control (jetzt im weiteren Text das Zeichen ^) PQ, ^PW, ^PE und ^PR. Diese vier Möglichkeiten sind zu wenig. Wir suchen daher noch nach weiteren und werden beim Menüpunkt <I> fündig. Unser Drucker besitzt nur ein Farbband, eine Umschaltungsmöglichkeit wird also nicht benötigt, so daß wir die Funktion ^PY anderweitig einsetzen können. ^PY hat zwei Funktionen, das erste ^PY schaltet einen Vorgang ein, das zweite ^PY schaltet wieder zurück.

Die Drucksteuerbefehle ^PA und ^PN, die wir unter dem Menüpunkt <Q> finden, dienen normalerweise der Steuerung der Zeichenbreite. Nach meiner Auffassung können wir diese Funktion besser nutzen, wir werden hier also eine Änderung vornehmen. Unter der Menüziffer <P> finden wir noch die Drucksteuerbefehle ^PT (Hochstellen) und ^PV (Tiefstellen). Wir werden diese Befehle für die Druckermöglichkeiten Sub- und Superscript nutzen.

Fangen wir also mit der Neuinstallation bei den Sequenzen für Farbbandauswahl an. Nach dem Installationsmenü für Drucker wählen wir den Buchstaben <I>. Kurz darauf werden wir gefragt, welche Sequenz beim ersten ^PY zum Drucker geschickt werden soll. Die nächste Zeile zeigt an, ob hier bereits Steuersequenzen installiert sind, wenn ja, welche, oder ob noch gar keine Steuerbefehle vorliegen (empty = leer). Im letzten Satz der Bildschirmanzeige werden wir angewiesen, bei beabsichtigten Änderungn die Taste <C> zu drücken. Wenn wir keine Änderungen wünschen, wäre mit <RETURN> weiterzugehen.

Da wir für das erste PY eine neue Sequenz wollen, müssen wir also <C> drücken. Es erscheint ein Bildschirm voller englischer Anweisungen, die wir aber bis auf den letzten Satz nicht weiter beachten brauchen. Der letzte Satz sagt uns, wieviel Eingaben möglich sind. Im vorliegenden Falle wird eine »4« angezeigt. Ganz unten sehen wir nebeneinander zwei Anzeigen:

links: »Current value« = gegenwärtiger Wert
rechts daneben: »New value« = neuer Wert.

Mit dem ersten ^PY wollen wir von der normalen Schriftbreite (80 Zeichen je Zeile) auf Engschrift (136 Zeichen je Zeile) umschalten. Die Eingabe der Steuersequenzen muß bei unserer Installationsversion hexadezimal erfolgen. Alle im folgenden benutzten Sequenzen finden wir im »Parallel-Steckmodul-Handbuch, Anhang D« zum Star NL-10. Aus dem

*** INSTALLATIONSMENÜ FÜR DRUCKER ***

A Automatische Installation für Spezialdrucker
B Automatische Installation für Standarddrucker

Alle Drucker	Nur Spezialdrucker
C Name des Druckers	I Farbbandauswahl
D Initialisierung	J Vertikale Bewegung (des Papiers)
E Überdrucken	K Horizontale Bewegung (des Druckkopfes)
F Fettdruck	L Druckmodi
G Protokollauswahl	M Phantomzeichen
H Treiberauswahl	
Nur Standdardrucker	**Optionen**
N Wagenrücklauf/Zeilenvorschub	O Benutzereigene Funktionen
X Zurück zum Installationsmenü	P Papiertransport
	Q Zeichenbreite

Drücken Sie den Buchstaben Ihrer Wahl (A - Q/X)

Bild 3. Übersetztes Druckerinstallationsmenü

MicroPro WordStar release 3.00 - 2.0 serial WS PC 128 M&T
Copyright (C) 1981 MicroPro International Corporation

Commodore 128 Personal Computer
Drucker Star NL-10, Inst. 1.0
No Communications protocol
Primary list device

Bild 4. »Normales« Titelbild beim Laden von Wordstar 3.0

Druckerhandbuch ergibt sich für Engschrift die Sequenz 1B (erster Wert) und OF (zweiter Wert). Da vier Eingabemöglichkeiten zur Verfügung stehen, setzen wir für die erste Möglichkeit 00 (jeweils weiter mit <RETURN>), für die zweite ebenfalls 00, für die dritte 1B und für die vierte OF. Nach dem letzten <RETURN> erhalten wir die Meldung, daß die neu eingegebene Steuersequenz jetzt lautet: 0h 0h 1Bh Fh. Wir sehen, daß von unserer Eingabe die erste 0 weggefallen ist, hinter jeden Wert wurde automatisch ein h für »hexadezimal« gestellt. Für unsere weitere Installation ist das »h« aber ohne Bedeutung. Falls wir jetzt feststellen, daß wir uns vertan haben, gehen wir mit Tastendruck <N> an den Anfang der Installation für das erste ^PY zurück. Entspricht die neue Sequenz aber unserer Eingabe, beantworten wir die folgende Sicherheitsabfrage mit <RETURN>. Damit ist die Eingabe abgeschlossen und das Installationsprogramm fragt jetzt, welche Steuersequenz das zweite ^PY auslösen soll.

Definition der Steuersequenzen

Ein Zurücksetzen der Engschrift zum normalen Zeichenabstand wird durch hexadezimal 12 ausgelöst. Wir arbeiten uns jetzt wieder genau wie beim ersten ^PY durch das Menü und geben als Wert 00 00 1B 12 ein. Mit dem nächsten <RETURN> ist die Installation des Druckermenüpunktes <I> abgeschlossen, auf dem Bildschirm erscheint wieder das Installationsmenü für Drucker. Wenden wir uns jetzt den Steuerbefehlen ^PA und ^PN zu, die wir im Druckerinstallationsmenü unter dem Menüpunkt <Q> finden. Mit diesen beiden Wordstar-3.0-Befehlen wollen wir softwaremäßig die NLQ-Schrift ein- beziehungsweise ausschalten. Drücken wir also die Taste <Q> und verfahren wie beim ersten ^PY beschrieben. Die Steuersequenz, die wir für »NLQ-Schrift ein« auf ^PA installieren wollen, lautet 00 1B 78 01, auf ^PN installieren wir »NLQ-Schrift aus« mit 00 1B 78 00.

Es übt sich und wird sicher immer einfacher. Wir gehen also jetzt weiter zu Menüpunkt P, um hier die Befehle ^PT und ^PV mit neuen Steuersequenzen zu belegen.

Die bisherigen Wordstar-3.0-Befehle ^PT und ^PV (Hoch- und Tiefstellen) funktionieren mit Standard-Matrixdruckern nur, wenn mit doppeltem Zeilenabstand geschrieben wird. Das will man normalerweise aber nicht. Bei einfachem Zeilenabstand wird der Befehl ignoriert. Da der NL-10 viele Möglichkeiten hat, lassen wir uns hier etwas einfallen. Wir verwenden Superscript und Subscript für Hoch- beziehungsweise Tiefstellen. Superscript hat als Steuersequenz 00 1B 53 00, Subscript hat die Sequenz 00 1B 53 01.

Nach Drücken des Menüpunktes <P> verfahren wir auch hier nach dem uns schon bekannten System. Mit diesen beiden neu installierten Sequenzen können wir aber nur Super- oder Subscript einschalten. Das genügt uns aber nicht, wir müssen danach ja wieder ausschalten können. Das tun wir mit dem Befehl ^PET. Dieser Befehl wird weiter unten noch erläutert werden. Wenn wir zum Beispiel die Angabe »m²« schreiben wollen, so müssen wir folgende Zeichen eingeben: m^PT2^PET. Ähnlich wäre bei Tiefstellen mit Subscript zu verfahren. Beim Ausdruck unter NLQ werden die hochbeziehungsweise tiefgestellten Zeichen allerdings in Normalschrift gedruckt.

Nun kommen wir zu den eigentlichen »Benutzereigenen Funktionen«, die wir unter dem Menüpunkt <O> installieren können. Hier gibt es vier Befehle, die auch in der nachstehenden Reihenfolge zu installieren sind: ^PQ, ^PW, ^PE und ^PR. Welche zusätzlichen Funktionen unseres Druckers möchten wir hier verwenden? Vorschlag: Sequenzen für doppelte und vierfache Schrifthöhe (speziell auf dem NL-10), für die mit der Funktion »Wiederausschalten« schon drei Befehle benötigt werden und schließlich eine »Universalsequenz«, auf die weiter unten noch näher eingegangen wird.

Die Steuersequenzen für den Installationsdurchgang <O> (Benutzereigene Funktionen), deren Installation nacheinander automatisch auf dem Bildschirm in der folgenden Reihenfolge aufgerufen wird, lauten:

für	^PQ 00 1B 68 1	Doppelte Schrifthöhe
für	^PW00 1B 68 2	Vierfache Schrifthöhe
für	^PE 00 00 00 1B	»Universalsequenz«
für	^PR 00 1B 68 0	von 2- beziehungsweise 4-facher Schrifthöhe zurück auf Normalschrift

Wir beenden die Installation dieses Menüpunktes wie unter ^PY beschrieben und »landen« wieder im Druckermenü. Die Installation könnte jetzt abgeschlossen werden, unser »Kind« sollte aber einen Namen bekommen. Beim Laden von Wordstar erscheint auf dem Bildschirm kurzfristig ein Eröffnungsbild, das für den Drucker normalerweise die Bezeichnung »Standarddrucker« enthält. Unter dem Menüpunkt <C> (Name des Druckers) kann für die Neuinstallation ein Name eventuell mit einer Installationsnummer wie zum Beispiel »Drucker Star NL-10, Inst. 1.0« eingegeben werden. Weitere zusätzliche Installationen können dann später mit 1.1, 1.2 und so weiter bezeichnet werden. Mit <C> rufen wir also den Installationspunkt »Name des Druckers« auf. Da wir eine Änderung wünschen, betätigen wir nochmals <C>. Jetzt können wir einen neuen Druckernamen eingeben. Die maximale Zeichenzahl für den Namen darf 34 Zeichen betragen. Damit der neue Name später im Eröffnungsbild zentriert erscheint, zählt man die Zeichenzahl des neuen Namens ab. Bei dem obigen Vorschlag ergeben sich 29 Zeichen inklusive der Leertasten, so daß wir vor dem Namen zwei Leerschritte eingeben. Nach abgeschlossener Eingabe und zweimaligem <RETURN> ist auch der neue Name installiert.

Damit wollen wir die Gesamtinstallation beenden. Mit Tastendruck <X> verlassen wir das Installationsmenü für Drucker, mit einem weiteren <X> verlassen wir auch das Hauptmenü. Aus dem jetzt erscheinenden Menü wählen wir <A>. Damit wird der gesamte Installationsvorgang abgeschlossen, die neue Datei WS.COM mit allen Änderungen wird gespeichert, schließlich kehrt der C128 zum CP/M-Prompt >A zurück.

Mit PIP kopieren wir die neu installierte Datei WS.COM auf die Wordstar-3.0-Arbeitsdiskette, wobei die alte Datei WS.COM überschrieben wird. (Achtung: Keinesfalls auf die Original-Wordstar-Diskette kopieren!)

Wenn jetzt Wordstar geladen wird, sollte ein Eröffnungsbild wie in Bild 4 erscheinen, jedoch mit Ihrem zuvor festgelegten Druckernamen und gegebenenfalls einer Versionsnummer.

Es zeigt uns, daß wir jetzt mit der neuen Version WS.COM arbeiten und die neuen Befehle der Tabelle 1 anwenden können.

Die »Universalsequenz«

Unter ^PE haben wir die Steuersequenz 00 00 00 1B installiert. 1B ist die hexadezimale Darstellung der sogenannten Escapesequenz <ESC> (in Basic CHR$ (27)). Mit ihr beginnen eine ganze Reihe von Steuercodes, mit denen wir

unseren NL-10 unter Wordstar veranlassen können, viele Dinge zu tun, die uns sonst mit diesem Textprogramm und einem Matrixdrucker nicht möglich wären. Zum Beispiel kann man mit Wordstar 3.0 und einem Standarddrucker normalerweise nur ein-, zweizeilig bis neunzeilig drucken (^OS 1-9). Punktbefehle wie zum Beispiel .LH funktionieren nur mit Typenraddruckern. Mit unserer »Universalsequenz« allerdings eröffnet sich fast eine Wunderwelt an neuen Möglichkeiten. Zum Beispiel gibt es einen Steuercode, der den Zeilenabstand des NL-10 in $^{n}/_{216}$-inch-Schritten bewirkt. Der Code lautet nach dem Handbuch CHR$ (27); "3"; n. Der Standardzeilenabstand unter Wordstar 3.0 beträgt bei einzeiliger Schreibweise 6 Zeilen / inch, $^{36}/_{212}$ inch wäre also einzeilig. 1 $^{1}/_{2}$zeilige Schreibweise läßt sich hieraus rechnerisch unschwer ableiten: (1-zeilig = 36, $^{1}/_{2}$-zeilig = 18, 1 $^{1}/_{2}$-zeilig = 36 + 18 = 54). Nochmals zurück zu unserem Code <ESC> "3" n. Für n muß man einsetzen, wieviele 216zehntel inch der Zeilenabstand betragen soll. Normalerweise müßte der Code für 1 $^{1}/_{2}$-zeilige Schreibweise lauten: <ESC> "3" 54. Da in diesen Steuercodes allerdings keine Dezimalzahlen, sondern nur ASCII-Codes stehen dürfen ("3" ist bereits das ASCII-Zeichen der Dezimalzahl 51), müssen wir noch das ASCII-Zeichen für die Dezimalzahl 54 suchen. Wir finden es in der Liste »ASCII-Codes und Vergleichsliste, Anhang B« des Steckmodul-Handbuches zum NL-10 oder jeder anderen ASCII-Vergleichsliste wie folgt: dezimal 54 = ASCII "6". Der richtige Code lautet also <ESC>;"3";"6". Unter Wordstar geben wir ^PE36 ein, da der Befehl für <ESC> von uns auf ^PE installiert wurde. Auf unserem Bildschirm erscheint ^E36. Bis zu einer weiteren Änderung wird unser Drucker jetzt 1 $^{1}/_{2}$-zeilig drucken. So läßt sich jede gewünschte Zwischengröße für die Zeilenabstände errechnen und verwirklichen.

Das alles liest sich viel komplizierter, als es in Wirklichkeit ist. Ein wenig Probieren und Nachdenken und die neue Drucker-Befehlsübersicht (Tabelle 1) machen diesen Vorgang völlig einfach. Fast alle Eigenschaften des NL-10, die sich nicht mit normalen Wordstar-3.0-Befehlen oder Punktbefehlen ansteuern lassen, sind mit unserer neuen »Universalsequenz« aktivierbar.

Der Einsatz der Neuinstallation

Der Einsatz kann natürlich nicht im vollen Umfang dargestellt werden, hier muß der Anwender probieren. Auf einige Besonderheiten sollte dennoch hingewiesen werden.

Wer hätte nicht gern die Möglichkeit, Artikel oder Abschnitte von Artikeln mit einem Großbuchstaben zu beginnen? (siehe beispielsweise die Artikel in der »64'er«). Hierzu muß man ein wenig nachdenken.

Bezeichnung	Eingabe	Bildschirmdarstellung
Schönschreibschrift (NLQ) ein	^PA	↑A
Schönschreibschrift (NLQ) aus	^PN	↑N
Superscript <hochgestellt> ein	^PT	↑T
Subscript <tiefgestellt> ein	^PV	↑V
Super-/Subscipt aus <siehe auch weiter unten>	^PET	↑ET
Kleinschrift <17 Zeichen/inch> ein	(erstes) ^PY	↑Y
Kleinschrift <17 Zeichen/inch> aus	(zweites) ^PY	↑Y
Doppelte Schrifthöhe	^PQ	↑Q
Vierfache Schrifthöhe	^PW	↑W
Doppelte oder vierfache Schrifthöhe aus (zurück zur vorherigen Schrifthöhe)	^PR	↑R
Universalsequenz (zusammen mit ein oder zwei weiteren Steuerzeichen der jetzt folgenden weiteren Auflistung)	^PE	↑E
Kursivschrift (Italic) ein	^PE4	↑E4
Kursivschrift (Italic) aus	^PE5	↑E5
Elite (12 Zeichen/inch) ein	^PEM	↑EM
Pica (Normal, 10 Zeichen/inch) ein	^PEP	↑EP
Breitdruck ein	^PEW1	↑EW1
Breitdruck aus	^PEW0	↑EW0
Proportionalschrift ein	^PEp1	↑Ep1
Proportionalschrift aus	^PEp0	↑Ep0
Super- bzw. Subscript aus	^PET	↑ET
Zeilenabstand		
6 Zeilen/inch (normal)	^PE2	↑E2
8 Zeilen/inch	^PE0	↑E0
10 Zeilen/inch	^PE1	↑E1
n Zeilen/inch (siehe auch Erklärung im Text)	^PE3n	↑E3n
Einmaliger Rücktransport des Papiers um $^{n}/_{216}$tel inch Berechnung zu n siehe Text	^PEjn	↑Ejn
Beispiel: Einmaliger Rücktransport des Papiers um 1 Zeile	^PEj$	↑Ej$

Tabelle 1. Die durch die Neuinstallation zusätzlich gewonnenen Druckbefehle

Der Einsatz der Neuinstallation

Normalerweise schreibt der Drucker nach Ausdruck eines Buchstabens mit doppelter oder vierfacher Höhe nach Zurückschalten auf Normalschrift etwas unterhalb der unteren Zeile, also sozusagen am Fuß des Großbuchstabens, weiter. Will man in die »Kopfzeile«, ist dies ohne weiteres mit der »Universalsequenz« ^PE möglich. Probieren Sie doch einfach einmal die folgende Zeichenkette am Beginn der nächsten Zeile:

^PWN^PR^PEjLUN

Als Ergebnis müßte der NL-10 »Nun« mit großem »N« ausgeben.

Nun können wir in der Kopfzeile weiterschreiben. Natürlich müssen wir jetzt in den folgenden zwei Zeilen am linken Rand jeweils vier Leerzeichen einfügen, da wir sonst den Großbuchstaben überschreiben. Der Rückschritt in die obere Zeile wurde durch die Sequenz ^PEjL (Rückschritt um $^{76}/_{216}$ inch, also etwas mehr als 2 Zeilen) bewirkt.

Natürlich muß man bei der späteren Verwendung der doppelten oder vierfachen Schrifthöhe mit Wordstar 3.0 »jonglieren«. Die Buchstaben werden trotz ihrer mehrfachen Breite und Höhe in der Statuszeile sowohl bei der Spalten- als auch bei der Seitenzahl einfach gezählt. Bei der Wordstar-3.0-Standardeinstellung mit 65 Zeichen/Zeile ist also die spätere Druckzeile bereits bei 32 doppeltbreiten beziehungsweise 16 vierfachbreiten Zeichen bis an den rechten Rand gefüllt. Eine Bildschirmdarstellung ist also, wie auch bei fast allen anderen Wordstar-3.0-Befehlen, nicht gegeben. Wenn man doppelte oder vierfache Zeichenhöhe über mehrere Zeilen schreiben möchte, sollte man nach jeder Schriftzeile sofort nach dem letzten Zeichen mit ^PR zurücksetzen und gegebenenfalls die neue Zeile wieder mit ^PQ oder ^PW beginnen, sonst multipliziert der NL-10 den links gesetzten Druckrand mit 2 oder 4 und beginnt dadurch die nächste Zeile nicht linksbündig, sondern irgendwo in der Mitte.

Was den großen Buchstaben recht ist, ist den kleinen billig. Mit ^PY^PT erhalten wir die kleinstmögliche Schrift auf unserem NL-10, die wir mit ^PY^PET beenden.

Auch die Italic-Breitschrift sieht hervorragend aus.

Sie wird mit dem Befehl ^PEW1^PE4....^PEWO^PE5 geschrieben.

Überhaupt ist eine Vielzahl von Kombinationen möglich, die man an dieser Stelle bei weitem nicht alle beschreiben kann. Probieren Sie doch ein wenig und Sie werden mit mir der Meinung sein, mit der Kombination

Commodore 128 PC - Star NL-10 - Wordstar 3.0

über ein vorzügliches und dennoch preiswertes Textverarbeitungssystem zu verfügen. (Uwe Steenbuck/bj)

Turbo-Pascal-Utility der Spitzenklasse

Erstmals stellen wir ein Utility zu Turbo-Pascal vor. Der Programmierer kann damit die Verwaltung der einzelnen Elemente seiner Programme organisieren.

Pascal-Programmierer, die sehr viele Programme schreiben, merken oft gar nicht, daß die eine oder andere Prozedur oder Funktion bereits des öfteren in anderen Programmen verwendet wurde. Ein großer Vorteil wäre die Katalogisierung der bereits verwendeten Programmteile. Doch wer macht sich schon die Mühe, jedesmal die einzelnen Schritte eines Programms genau zu protokollieren. Die Lösung dazu bietet das im folgenden beschriebene Utility (Listing 1), das eine Quelldatei wie eine Bibliothek behandelt.

Nach dem Compilieren des Programms unter Turbo-Pascal können Sie es am Quellcode sofort austesten. Dabei haben Sie verschiedene Möglichkeiten. Die Ausgabe kann entweder auf den Bildschirm oder auf Diskette gelenkt werden. Nehmen wir an, das Programm hat den Namen »LIB.COM«. Der Quellcode ist mit der Bezeichnung »LIB.PAS« auf derselben Diskette abgelegt. Soll die Ausgabe auf den Bildschirm erfolgen, geben Sie folgende Befehlszeile ein:

```
LIB LIB.PAS
```

Alle Funktionen und Prozeduren werden auf dem Bildschirm mit Kennbuchstabe (»F« für Funktion und »P« für Prozedur), programminternem Namen und der Bedeutung ausgegeben (Bild 1). In der letzten Spalte findet sich der zugehörige Name des Pascal-Programms, aus dem das Element stammt.

Natürlich kann die Tabelle auch auf eine Datei in einem beliebigen Laufwerk ausgegeben werden. Zur Veranschaulichung soll die Tabelle auf die Datei GESAMT.LIB in Laufwerk B ausgegeben werden. Dazu ist folgende Befehlszeile erforderlich:

```
LIB LIB.PAS B:GESAMT.LIB
```

Für die Eingabe in das Programm sind auch die sogenannten Joker-Zeichen (?,*) erlaubt. Damit können alle Pascal-Programme auf einer Diskette nach Funktionen und Prozeduren durchsucht werden. Somit befindet sich nach kurzer Zeit eine umfangreiche Bibliothek in einer einzigen Datei und das lästige Suchen hat für immer ein Ende. Einige Formalitäten müssen allerdings beachtet werden. Der Kopf einer jeden Funktion oder Prozedur muß als Kommentar in einem bestimmten System aufgebaut werden, damit er als solcher von LIB.COM erkannt wird (Bild 2). Dadurch wird zusätzlich die Lesbarkeit des Programms verbessert.

Wer gerne über Hochsprachen ins System vorstoßen möchte, findet im Listing einige Kniffe, mit denen das BDOS manipuliert wird. So wird beispielsweise der Datenpuffer (DMA) gesetzt und verändert oder der File-Control-Block (FCB) direkt mit Informationen versorgt.

Zwar wirkt die Länge des Listings abschreckend, doch das Abtippen wird sich lohnen. Jeder ernsthafte Programmierer

```
TYP! NAME                 ! FUNCTION                                ! MODUL
---+--------------------+-----------------------------------------+-------
 F !read_port           !Liest den Zustand eines I/O-Ports ein    !UTIL.PAS
   !                    !und ersetzt die vom PASCAL-Compiler      !
   !                    !falsch uebersetzte Anweisung             !
   !                    !var:=port[loc];                          !
   !                    !                                         !
 F !bcd_to_string       !wandelt eine in value enthaltene         !UTIL.PAS
   !                    !BCD-zahl nach ASCII um                   !
   !                    !                                         !
 F !set_clock           !setzt hardware-uhr in der cia 1          !UTIL.PAS
   !                    !                                         !
 P !read_clock          !Liest Hardware-Uhr der CIA1 achtung!     !UTIL.PAS
   !                    !procedure read_port wird benoetigt!      !
   !                    !                                         !
 P !cursor_control      !manipuliert den cursor des 80-zeichen    !UTIL.PAS
   !                    !schirmes                                 !
   !                    !                                         !
 P !append              !bewegt den dateizeiger zum ende einer    !UTIL.PAS
   !                    !textdatei danach ist nur noch schreiben  !
   !                    !moeglich entnommen aus 'computer         !
   !                    !persoenlich' 9/86                        !
```

Bild 1. Element-Tabelle nach Anwendung von LIB.COM auf ein kleineres Programm

```
{**************************************************************************}
{                                                                          }
{ FUNCTION : tab  (spalte:integer): string[80];                            }
{                                                                          }
{ FUNKTION : simuliert den fehlenden Tabulator-Befehl                      }
{                                                                          }
{ EINGABE  : spalte: Anzahl der Spalten, um die der Cursor bewegt wird     }
{                                                                          }
{ AUSGABE  : tab: blank line mit der Laenge der Spalte                     }
{                                                                          }
{**************************************************************************}
```

Bild 2. Für LIB.COM notwendiger Kopfaufbau

sollte sich bemühen, seine Quellcodes übersichtlich zu gestalten, um einen ständigen Überblick über die verwendeteten Funktionen und Prozeduren zu haben. Dieses Spiel kann man am Ende soweit treiben, daß ähnliche Programme nur noch aus bereits vorhandenen zusammengesetzt und leicht modifiziert werden. In diesem Sinne: Weiterhin viel und vor allem noch mehr Spaß mit Turbo-Pascal.

(Wolfgang Schröder/rf)

```
{*************************************************************************}
{                                                                         }
{ PROGRAMM        : lib.pas                                               }
{                                                                         }
{ AUTOR           : wolfgang schroeder                                    }
{                                                                         }
{ ERSTELLT AM     : 20.07.86                                              }
{                                                                         }
{ LETZTE KORREKTUR :                                                      }
{                                                                         }
{ COMPILER        : turbo pascal 3.0                                      }
{                                                                         }
{ HARDWARE        : commodore c128 oder anderer cp/m-rechner              }
{                                                                         }
{ FUNKTION        : erstellt eine uebersicht der in pascal-sourcen        }
{                   enthaltenen functions und procedures                  }
{                                                                         }
{*************************************************************************}

{$u+}

type
   fcb_type = record
                drive : byte;
                fname : array[0..10] of char;
                extend: byte;
              end;
   file_string = string[14];
   dma_type = array[0..127] of char;
   line = string[132];

const
   aufruf = 'Aufruf: lib infile [outfile]';
   blanks = '
   under_line = '------------------------------------------';

var
   infile, outfile: file_string;
   workfile: array[1..32] of file_string;
   in_var, out_var: text;
   out_info: boolean;
   answer: char;
   fcb   : fcb_type absolute $5c;
   dma   : dma_type;
   work_ind,index,result : integer;
   in_line,out_line: line;
   drive: string[2];

{*************************************************************************}
{                                                                         }
{ FUNCTION : tab (spalte:integer): string[80];                            }
{                                                                         }
{ FUNKTION : simuliert den fehlenden tabulator-befehl                     }
{                                                                         }
{ EINGABE  : spalte: anzahl der spalten, um die cursor gewegt wird        }
{                                                                         }
{ AUSGABE  : tab: blank line mit der laenge spalte                        }
{                                                                         }
{*************************************************************************}

function tab(spalte:integer): line;
const
  bl_line = '

var
  print: string[80];

begin
   if spalte>length(bl_line) then
      tab:= '*'
   else
      tab:= copy(bl_line,1,spalte);
end; {tab}

{*************************************************************************}
{                                                                         }
{ FUNCTION : check_input (var file1,file2:string[14]): boolean            }
{                                                                         }
{ FUNKTION : Holt eventl. vorhandene beim Programmaufruf angegebene       }
{            Parameter und max. 2 Filenamen ab                            }
{                                                                         }
{ EINGABE  : keine, da Parameterversorgung ueber ParamCount und ParamStr  }
{                                                                         }
{ AUSGABE  : false -> falsche Parametereingabe                            }
{            true  -> Parametereingabe ok                                 }
{                       infile: string[14] -> 1. filename                 }
{                       outfile:string[14] -> 2. filename                 }
{                                                                         }
{*************************************************************************}

function CHECK_INPUT (var file1,file2:file_string): boolean;

begin
   check_input:= false;
   file1:= '';
   file2:= '';

   case ParamCount of
   1:     begin
            file1:= ParamStr(1);
            check_input:= true
          end;
   2:     begin
            file1:= ParamStr(1);
            file2:= ParamStr(2);
            check_input:= true;
          end;
     else writeln (aufruf);
     end; {case}

end; {check_input}
```

Listing 1. Der Quellcode zu LIB.PAS

```
{*************************************************************************}
{                                                                         }
{ FUNCTION : open_output (outfile:string[14]): boolean;                   }
{                                                                         }
{ FUNKTION : oeffnet ausgabedatei fuer textausgabe                        }
{                                                                         }
{ EINGABE  : outfile: name der ausgabedatei                               }
{            append : true  -> ausgabe an bestehende datei anfuegen       }
{                     false -> ausgabe auf neue datei                     }
{                                                                         }
{ AUSGABE  : true  -> ausgabedatei geoeffnet                              }
{            false -> ausgabedatei nicht geoeffnet                        }
{                                                                         }
{*************************************************************************}

function OPEN_OUTPUT(outfile:file_string): boolean;

begin
  open_output:= false;
  assign (out_var,outfile);

  {$i-} reset (out_var) {$i+};
  if IOresult = 0 then begin
     close (out_var);
     gotoxy(3,6);
     write ('Ausgabedatei existiert bereits! Ueberschreiben [j/n] ? ');
     readln (answer);
     if answer = 'j' then begin
        erase (out_var);
        rewrite (out_var);
        open_output:= true;
     end;
  end
  else begin
    rewrite (out_var);
    open_output:= true;
  end;
end; {open_output}

{*************************************************************************}
{                                                                         }
{ FUNCTION : fname_to_fcb (fname: string[14]): boolean                    }
{ -                                                                       }
{ FUNKTION : kopiert den angegebenen filename unter beruecksichtigung     }
{            der fcb-syntax in den standart fcb bei $5c. der filename     }
{            darf auch die jokerzeichen '?' und '*' enthalten.            }
{                                                                         }
{ EINGABE  : fname : in den fcb zu kopierenden filenamen                  }
{                                                                         }
{ AUSGABE  : true  -> filename ok                                         }
{                     fcb enthaelt konvertierten filenamen                }
{            false -> syntaxfehler im filenamen                           }
{                                                                         }
{*************************************************************************}

function FNAME_TO_FCB (fname: file_string): boolean;

var
   ind_start,index1, index2: integer;

begin
   fname_to_fcb:= true;
   index2:= 0;

   if fname[2]=':'  then begin
      fcb.drive:= ord(fname[1])-64;
      ind_start:= 3;
      end
   else begin
      fcb.drive:= 0;
      ind_start:= 1;
   end;

   for index1:= ind_start to length(fname) do begin
      case fname[index1] of
      'A'..'Z',
      '0'..'9': begin
                fcb.fname[index2]:= fname[index1];
                index2:= succ(index2);
                end;
      '.':      begin
                if index2<8 then
                   for index2:= index2 to 8 do
                      fcb.fname[index2]:= ' ';
                end;
      '?':      begin
                fcb.fname[index2]:= '?';
                index2:= succ(index2);
                end;
      '*':      begin
                if index2<8 then
                   for index2:= index2 to 8 do
                      fcb.fname[index2]:= '?'
                else
                   for index2:= index2 to 10 do
                      fcb.fname[index2]:= '?';
                end;
      else      begin
                writeln ('falscher dateinamen fuer inputfile');
                fname_to_fcb:= false;
                end;
      end; {case}
      fcb.extend:= 0;
   end;

end; {fcb_to_fname}

{*************************************************************************}
{                                                                         }
{ PROCEDURE: dma_to_fname (var filename:string[14];                       }
{                          dma: array[0..127] of char;                    }
{                          dma_pos: integer);                             }
{                                                                         }
{ FUNKTION : setzt einen directory eintrag im 'dma' in einen fuer         }
{            dateizugriffe gueltigen filename um                          }
{                                                                         }
{ EINGABE  : dma     : von einer bdos-funktion ausgefuellter dma-bereich  }
{            dma_pos: startposition im dma fuer den fileeintrag           }
{                                                                         }
{ AUSGABE  : filename: konvertiert fileeintrag fuer dateizugriffe         }
{                                                                         }
{*************************************************************************}
```

```
procedure DMA_TO_FNAME (var filename: file_string;
                          dma: dma_type;
                          dma_pos: integer);

var
  start: integer;

begin
   filename:= copy (dma,dma_pos+2,8); {dateiname holen}
   start:= pos (' ',filename);         {blanks entfernen}
   if start>0 then
      delete (filename,start,5);
   filename:= concat (filename,'.',copy (dma,dma_pos+10,3) ); {ext. dazu}
end; {dma_to_fname}

{**********************************************************************}
{                                                                      }
{ FUNCTION : search_header (var in_file: text;                         }
{                           var in_line: string[132]): boolean;        }
{                                                                      }
{ FUNKTION : durchsucht die angegebene textdatei nach functions- und   }
{            procedure-headern                                         }
{                                                                      }
{ EINGABE  : in_file: dateibeschreibung der eingabedatei               }
{                                                                      }
{ AUSGABE  : true  -> header gefunden                                  }
{                     in_line: zeile mit headerbeginn                  }
{            false -> keinen header gefunden                           }
{                                                                      }
{**********************************************************************}

function SEARCH_HEADER (var in_file: text;
                        var in_line:line): boolean;

begin
  search_header:= false;
  while not eof(in_file) do begin
     readln (in_file,in_line);
     if ((pos('{ FUNCTION :',in_line)=1) or
         (pos('{ PROCEDURE:',in_line)=1)) then begin
         search_header:= true;
         exit;
     end;
  end; {while}

end; {search_header}

{**********************************************************************}
{                                                                      }
{ PROCEDURE: extract_info (var in_file,out_file:text; out_info:boolean;  }
{                          in_line:string[132]; workfile:string[14]);  }
{                                                                      }
{ FUNKTION : extrahiert ausgabeinformation ueber die uebergebene function }
{            oder procedure                                            }
{                                                                      }
{ EINGABE  : in_file : dateibeschreibung der eingabedatei              }
{            out_file: dateibeschreibung der ausgabedatei              }
{            out_info: ausgabe auf datei(true) oder bildschirm(false); }
{            in_line : zeile mit function- bzw. procedurenamen         }
{            workfile: name der eingabedatei                           }
{                                                                      }
{ AUSGABE  : extrahierte information                                   }
{                                                                      }
{**********************************************************************}

procedure EXTRACT_INFO (var in_file,out_file:text; out_info:boolean;
                        var in_line: line;workfile:file_string);

type
   func_type = array[0..9] of string[80];
var
   out_line: string[80];
   func_proc: char;
   fp_name: string[20];
   func_desc: func_type;
   index: integer;
   loop, blank_pos:integer;


{**********************************************************************}
{                                                                      }
{ FUNCTION : split (var in_line: func_type; max_line:integer;          }
{                   var index: integer; max_length: integer): line     }
{                                                                      }
{ FUNKTION : reformatiert eine textzeile auf eine andere laenge        }
{                                                                      }
{ EINGABE  : in_line = array[] of string[]                             }
{                      original text (wird veraendert!)                }
{            max_line = anzahl der zu reformatierende zeilen in in_line  }
{            max_length = zeilenlaenge des neuen textes                }
{                                                                      }
{ AUSGABE  : split = string[]   reformatierte zeile                    }
{            index = zeilennummer der zuletzt reformatierten zeile     }
{                                                                      }
{**********************************************************************}

function split(var in_line: func_type; max_line:integer;
               var index:integer; max_length:integer): line;

var temp1,temp2: string[80];
    bl_pos:integer;

begin

temp2:='';

repeat
   bl_pos:= pos(' ',in_line[index]);
   if bl_pos=0 then
      temp1:= in_line[index]+' '
   else
      temp1:= copy(in_line[index],1,bl_pos);
   if length(temp1+temp2)>max_length then begin
      split:=temp2;
      exit
   end
   else begin
      temp2:= temp2+temp1;
      delete (in_line[index],1,bl_pos);
      if bl_pos=0 then
         index:= succ(index);
   end;
until (index>max_line);
split:= temp2;
end;  {split}

begin  { of extract_info}
   if pos('{ FUNCTION :',in_line)=1 then
      func_proc:= 'F'
   else
      func_proc:= 'P';
   index:= pos('(',in_line);
   if ((index=0) or (index>34)) then
      fp_name:= copy(in_line,14,20)
   else
      fp_name:= copy(in_line,14,index-14);

   repeat
      readln (in_file,in_line);
   until (pos('{ FUNKTION :',in_line)=1);

   delete (in_line,1,12);   {kommentarzeichen entfernen}
   index:= -1;

   repeat
      delete (in_line,1,1);
      while in_line[1]=' ' do   {blanks am zeilenbeginn entfernen}
         delete (in_line,1,1);

      if in_line<>'}' then begin
         delete (in_line,length(in_line),1);
         while in_line[length(in_line)]=' ' do
            delete (in_line,length(in_line),1);

         index:= succ(index);
         func_desc[index]:= in_line;

         readln (in_file,in_line);
      end;
   until ((in_line='}') or (index=9));

   loop:= 0;

   repeat
      out_line:=' '+func_proc+' !'+fp_name;
      out_line:= copy(out_line+blanks,1,23);
      out_line:= out_line+'!'+split(func_desc,index,loop,40);
      out_line:= copy(out_line+blanks,1,65);
      out_line:= out_line+'!'+workfile;
      if out_info then
         writeln (out_var,out_line)
      else
         writeln (out_line);
      fp_name:= '';
      func_proc:= ' ';
      workfile:= '';
   until (loop>index);

end; {extract_info}

{********************}
{** hauptprogramm  **}
{********************}

begin
   clrscr;
   if CHECK_INPUT(infile,outfile) then begin
      if length(outfile) <> 0 then begin
         gotoxy (3,3);
         Write ('Ausgabedatei: ',outfile);
         if not(OPEN_OUTPUT(outfile)) then
            exit
         else
            out_info:= true;
            gotoxy (3,9);
            write ('Datei: ',outfile,' wird geschrieben');
      end
      else
         out_info:= false;
      if infile[2] = ':' then
         drive:= copy (infile,1,2)
      else
         drive:= '';
      if FNAME_TO_FCB(infile) then begin
         work_ind:= 1;
         bdos (26,addr(dma));    {set dma address}
         result:= bdos (17,addr(fcb));  {search for first}
         while ((result<>255) and (work_ind<33)) do begin
            index:= result shl 5;  {startadresse des directory entry}
            DMA_TO_FNAME(workfile[work_ind],dma,index);
            workfile[work_ind]:= drive+workfile[work_ind]; {laufwerk dazu}
            work_ind:= succ(work_ind);
            result:= bdos (18,addr(fcb)); {search for next}
        end; {while}
        if result<>255 then begin
           writeln ('es koennen nicht mehr als 32 files bearbeitet werden!');
           exit;
        end;
        work_ind:= work_ind-1;
        if work_ind=0 then begin
           writeln ('kein inputfile gefunden!');
           exit;
        end;
        if out_info then begin
           writeln (out_var,'TYP! NAME',tab(14),'! FUNCTION',tab(32),'! MODUL');
           writeln (out_var,'---+',copy(under_line,1,19),'+',
                 copy(under_line,1,41),'+------');
       end
       else begin
          writeln ('TYP! NAME',tab(14),'! FUNCTION',tab(32),'! MODUL');
          writeln ('---+',copy(under_line,1,19),'+',
                copy(under_line,1,41),'+------');
       end;
       for index:= 1 to work_ind do begin
          assign (in_var,workfile[index]);
          reset (in_var);
          while SEARCH_HEADER (in_var,in_line) do begin
             EXTRACT_INFO(in_var,out_var,out_info,in_line,workfile[index]);
             if out_info then
                writeln (out_var,tab(3),'!',tab(19),'!',tab(41),'!')
             else
                writeln (tab(3),'!',tab(19),'!',tab(41),'!');
          end;
          close (in_var);
        end;
     end;
     if out_info then
        close (out_var);
  end;
end.
```

Listing 1. Der Quellcode zu LIB.PAS (Schluß)

Hinweise zum Abtippen

Sie haben kein Problem mehr mit dem Abtippen von Basic-Programmen, wenn Sie die folgenden Hinweise zum Abtippen beachten.

Unsere Basic-Listings enthalten keine Steuerzeichen mehr. Diese werden ersetzt durch Klartext und stehen zwischen geschweiften Klammern. Deshalb sind weder die Klammern noch was dazwischen steht, abzutippen, sondern die in Tabelle 1 aufgeführten Tasten zu drücken. Auf Ihrem Bildschirm erhalten Sie dann wieder die entsprechenden Grafikzeichen (siehe Bild 1 und 2).

Alle Grafikzeichen werden ebenfalls ersetzt durch unterstrichene oder überstrichene Großbuchstaben.

Unterstrichene Buchstaben bedeuten, daß Sie die <SHIFT>-Taste und den angegebenen Buchstaben drücken müssen, überstrichene jedoch die <CBM>-Taste mit dem Buchstaben.

Auch hier erhalten Sie am Bildschirm das entsprechende Grafikzeichen und nicht etwa das im Listing erkennbare Zeichen (siehe Bild 1 und 2).

Die Leerzeichen zwischen den einzelnen Basic-Befehlen können beim Abtippen entfallen. Dies ist besonders bei speicherkritischen Programmen wichtig.

Ebenso müssen Zeilen, die mehr als 80 Zeichen pro Zeile enthalten, mit den bekannten Abkürzungen für die Basic-Befehle (siehe auch das Handbuch zum Computer, im Anhang) eingegeben werden. (gk)

```
5 PRINT CHR$(14)                              <242>
10 PRINT"{CLR}"                               <254>
20 PRINT"Q****************************W"      <130>
30 PRINT"{4DOWN,2SPACE}TEST{SPACE,BLUE,6SP
   ACE}"                                      <022>
40 PRINT"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"      <108>

© 64'er
```

Bild 1. So könnte ein Teil eines Listings abgedruckt sein. In Zeile 10 müssen Sie nach den Anführungsstrichen die <CLEAR+HOME>-Taste drücken und nicht die Klammern mit dem Wort CLR. In Zeile 20 drücken Sie nach den Anführungsstrichen die <CBM>-Taste und den Buchstaben Q, gefolgt von mehreren <SHIFT>- und STERN-Tasten und zum Schluß die Commodore-Taste und den Buchstaben W. In Zeile 30 ist es viermal die Cursor-nach-unten-Taste, gefolgt von zweimal die Leertaste, dann <SHIFT> und T und normal EST, zum Schluß noch einmal die Leertaste, die Farbtaste Blau (<CONTROL> und 7) und sechsmal die Leertaste. Zeile 40 besteht lediglich aus mehreren Grafikzeichen, die mit der <CBM>-Taste und <B> erzeugt werden.

```
5 PRINTCHR$(14)
10 PRINT"□"
20 PRINT"├──────────────────────────────┤"
30 PRINT"▓▓▓▓  │EST ▣      "
40 PRINT"▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚▚"
```

Bild 2. Auf dem Bildschirm oder Ihrem Drucker sieht das Listing (Bild 1) so aus

{CTRL}	steht für Control-Taste, so bedeutet {CTRL-A}, daß Sie die Control-Taste und die Taste »A« drücken müssen. Im folgenden steht:
{DOWN}	Taste neben rechtem Shift, Cursor unten
{UP}	Shift-Taste & Taste neben rechtem Shift; Cursor hoch
{CLR}	Shift-Taste & 2. Taste ganz rechts oben
{INST}	Shift-Taste & Taste ganz rechts oben
{HOME}	2. Taste von ganz rechts oben
{DEL}	Taste ganz rechts oben
{RIGHT}	Taste ganz rechts unten
{LEFT}	Shift-Taste & Taste unten rechts
{SPACE}	Leertaste, Hinweis: {13 SPACE} bedeutet 13mal die Leertaste drücken
{SHFT-SPCE}	Shift-Taste & Leertaste
{F1}	grauer Tastenblock rechts
{F3}	grauer Tastenblock rechts
{F5}	grauer Tastenblock rechts
{F7}	grauer Tastenblock rechts
{F2}	grauer Tastenblock rechts & Shift
{F4}	grauer Tastenblock rechts & Shift
{F6}	grauer Tastenblock rechts & Shift
{F8}	grauer Tastenblock rechts & Shift
{RETURN}	Shift-Taste & Return
{CTRL+I}	TAB-Taste

{CTRL+J}	Line Feed
{CTRL}	Control-Taste
{BLACK}	Control-Taste & 1
{WHITE}	Control-Taste & 2
{RED}	Control-Taste & 3
{CYAN}	Control-Taste & 4
{PURPLE}	Control-Taste & 5
{GREEN}	Control-Taste & 6
{BLUE}	Control-Taste & 7
{YELLOW}	Control-Taste & 8
{RVSON}	Control-Taste & 9
{RVOFF}	Control-Taste & 0
{ORANGE}	Commodore-Taste & 1
{BROWN}	Commodore-Taste & 2
{LIG.RED}	Commodore-Taste & 3
{GREY 1}	Commodore-Taste & 4
{GREY 2}	Commodore-Taste & 5
{LIG.GREEN}	Commodore-Taste & 6
{LIG.BLUE}	Commodore-Taste & 7
{GREY 3}	Commodore-Taste & 8

Wenn Sie sich erst einmal an die in Klartext geschriebenen Steuerzeichen gewöhnt haben, werden Sie den Vorteil dieser Schreibweise erkennen. Der zu dem jeweiligen Steuerzeichen gehörende Klartext ist so verfaßt, daß Sie leicht die Taste beziehungsweise die Tastenkombination finden, die Sie drücken müssen.

Tabelle 1. Die Steuerbefehle für den C64/C128 im Klartext

MSE – Abtippen sicher und leichtgemacht

Der MSE ist ein leicht zu bedienendes Hilfsmittel bei der Eingabe von Maschinensprache-Programmen im C 64-Modus.

Der MSE verringert die Tipparbeit um ein Drittel und schließt Fehleingaben vollkommen aus. Außerdem können Sie die Werte blind eingeben, ohne andauernd auf den Bildschirm schauen zu müssen. Dies wird durch akustische Meldungen realisiert.

MSE ist ein Maschinensprache-Editor, mit dem ein Vertippen ausgeschlossen ist. Eine abgetippte Zeile wird nur angenommen, wenn sie richtig ist. Eine Checksumme am Ende jeder Zeile prüft, ob die richtigen Werte in der richtigen Zeile an der richtigen Stelle stehen. Wenn nicht, ertönt ein Warnsignal, und man beseitigt den Fehler.

War die Zeile korrekt, erklingt ein Gong, und die nächste Zeilennummer wird ausgegeben. Damit ist also auch »blindes« Eintippen möglich; Sie können sich voll auf den Text konzentrieren.

So arbeitet man mit MSE

Aktivieren Sie den C 64-Modus, laden und starten Sie MSE. Zuerst werden der Programmname und die Start- und Endadresse erfragt. **Diese Angaben entnehmen Sie dem Kopf des jeweiligen abgedruckten Listings.** Der MSE meldet sich dann mit der Zeilennummer der ersten Zeile.

Wenn Sie die Zeile richtig eingegeben haben, erscheint die nächste Zeilennummer und so weiter bis zum Ende. Zum Schluß wird das fertige Programm mit <CTRL+S> auf Diskette oder Kassette gespeichert. Dazu sind keine weiteren Angaben mehr erforderlich. Das Programm kann dann ganz normal wieder geladen und gestartet werden. Wenn Sie nicht alles auf einmal tippen wollen, können Sie jederzeit unterbrechen und den eingetippten Teil mit <CTRL+S> speichern. Wollen Sie weiterarbeiten, laden und starten Sie MSE wieder.

Geben Sie auf die Frage nach der Startadresse aber jetzt <L> ein, um Ihr Teilprogramm zu laden. Jetzt können Sie mit <CTRL+N> die Adresse eingeben, an der Sie weitertippen müssen. Wenn Sie sich nicht gemerkt haben, wie weit Sie gekommen sind, geben Sie nach dem Laden <CTRL+M> ein.

Auf die Frage nach der Startadresse antworten Sie mit der Anfangsadresse, die links in der Kopfzeile auf dem Bildschirm steht. Nun wird Ihr Programm aufgelistet. Mit <SPACE> wird das Listen fortgesetzt, mit <STOP> abgebrochen. Das Ende Ihres Programmteils erkennen Sie sehr einfach daran, daß nur noch der Wert »AA« in der Zeile steht. Die Adresse dieser Zeile müssen Sie anschließend mit <CTRL+N> eingeben. Das Programm ist nur mit <RUN/STOP+RESTORE> zu verlassen. Speichern Sie aber vorher unbedingt immer Ihren Text.

Hinweise zum Abtippen

Vor dem Abtippen oder späteren Wiederladen des MSE-Laders müssen Sie unbedingt folgende Zeile eingeben:

POKE 43,1: POKE 44,32: POKE 8192,0: NEW

Den MSE-Lader brauchen Sie nur einmal. Nach erfolgreichem Abtippen und Starten mit RUN geht der Lader verloren, und es wird das endgültige Programm MSE V1.0 erzeugt. So gehen Sie vor:

Starten Sie das Programm mit RUN. Fehlerhafte Zeilen werden angezeigt und müssen korrigiert werden, bis der Lader zum »READY« durchläuft. Jetzt müssen Sie das fertige MSE-Programm speichern. Dazu brauchen Sie nur <RETURN> zu drücken, weil die erforderlichen Angaben schon auf dem Bildschirm stehen. (Kassettenbesitzer müssen in Zeile 343 die letzte Zahl in »1« abändern.) Ab jetzt können Sie »MSE V1.0« direkt, also ohne den DATA-Lader, benutzen. MSE V1.0 wird ganz normal mit »,8« geladen (keine POKEs notwendig).

(N. Mann/D. Weineck/gk)

MSE-Befehle:

<DEL>	löscht die letzte Eingabe.
<CTRL+S>	speichert das eingetippte Programm ab.
<L> oder <CTRL+L>	lädt ein Programm. Start- und Endadresse werden automatisch ermittelt.
<CTRL+M>	listet den Speicherinhalt. Abbruch mit STOP-Taste, weiter mit Leertaste.
<CTRL+N>	erlaubt die Eingabe einer neuen Adresse zum Weitertippen.
<CTRL+P>	gibt ein MSE-Listing auf dem Drucker aus.

```
100 REM ****************************      <091>
110 REM *                          *      <159>
120 REM *        M S E   LADER     *      <206>
130 REM *                          *      <179>
220 REM ****************************      <211>
230 REM                                   <036>
240 DIM H(75): FOR I=0 TO 9               <113>
250 H(48+I)=I: H(65+I)=I+10:NEXT          <041>
260 FOR I=2048 TO 3755 : READ A$          <198>
270 H=ASC(LEFT$(A$,1)):L=ASC(RIGHT$(A$,1)) <199>
280 D=H(H)*16+H(L):S=S+D:POKE I,D         <219>
290 A=A+1:IF A<20 THEN NEXT:A=-1          <141>
300 PRINT " ZEILE:";1000+Z;               <011>
310 READ V :Z=Z+1:IF V=S THEN 330         <218>
320 PRINT"PRUEFSUMMENFEHLER !":STOP       <138>
330 IF A<0 THEN 341                       <221>
340 S=0:A=0:PRINT:NEXT                    <046>
341 PRINT"{CLR}PO43,1:PO44,8:PO45,172:PO46
    ,14                                   <010>
342 POKE 631,19:POKE 632,13:POKE 633,13:PO
    KE 198,3                              <249>
343 PRINT"{3DOWN}SAVE"CHR$(34)"MSE V1.0"CH
    R$(34)",8                             <171>
344 END                                   <092>
1000 DATA 00,0B,08,0A,00,9E,32,30,36,31,00
     ,00,00,A2,08,A9,36,85,A4,A9, 1247    <119>
1001 DATA 08,85,A5,A9,00,85,A6,A9,B0,85,A7
     ,A0,00,B1,A4,91,A6,C8,D0,F9, 2888    <054>
1002 DATA E6,A5,E6,A7,CA,D0,F2,A9,36,85,01
     ,4C,00,B0,20,D1,B1,A9,06,8D, 2787    <144>
1003 DATA 21,D0,A9,03,8D,20,D0,8D,86,02,A0
     ,B3,A9,74,20,FF,B1,A0,B3,A9, 2667    <237>
1004 DATA B9,20,FF,B1,A0,00,20,CF,FF,99,01
     ,02,C8,C9,0D,D0,F5,88,F0,D2, 2912    <217>
1005 DATA C0,0F,90,02,A0,0E,8C,00,02,20,EA
     ,B1,A0,B3,A9,CF,20,FF,B1,20, 2323    <013>
1006 DATA 8E,B4,85,FC,85,62,20,8E,B4,85,FB
     ,85,61,20,A7,B4,D0,20,A0,B3, 2864    <199>
1007 DATA A9,E5,20,FF,B1,20,8E,B4,85,60,20
     ,8E,B4,85,5F,20,A7,B4,D0,0A, 2624    <091>
```

Der MSE zum bequemen Abtippen von Maschinenprogrammen

```
1008 DATA A5,61,C5,5F,A5,62,E5,60,90,06,20
     ,43,B3,4C,3A,B0,A9,AA,A0,00, 2379     <167>
1009 DATA 91,FB,E6,FB,D0,02,E6,FC,20,3F,B2
     ,90,EF,4C,FB,B4,A2,02,86,58, 3118     <152>
1010 DATA A9,A6,A0,9D,20,F2,B1,20,E4,FF,F0
     ,FB,C9,30,90,0C,C9,47,B0,08, 2970     <231>
1011 DATA C9,3A,90,0B,C9,41,B0,07,C9,14,D0
     ,0F,4C,0B,B1,20,D2,FF,A6,58, 2322     <121>
1012 DATA 95,F7,C6,58,D0,D2,60,AE,8D,02,F0
     ,26,C9,0C,D0,03,4C,0B,B6,C9, 2685     <057>
1013 DATA 13,D0,03,4C,8B,B5,C9,0D,D0,03,4C
     ,BA,B4,C9,10,D0,03,4C,68,B5, 2282     <225>
1014 DATA C9,0E,D0,06,20,5F,B4,4C,64,B1,4C
     ,92,B0,A5,F9,20,02,B1,0A,0A, 2132     <208>
1015 DATA 0A,0A,85,F9,A5,F8,20,02,B1,05,F9
     ,60,C9,3A,90,02,69,08,29,0F, 1950     <092>
1016 DATA 60,A6,59,E0,08,90,1F,A6,58,E0,02
     ,B0,06,20,D2,FF,4C,8E,B0,C6, 2509     <188>
1017 DATA 59,A0,14,A9,92,20,F2,B1,CA,D0,FA
     ,84,57,68,68,4C,8B,B1,A6,D3, 2891     <197>
1018 DATA E0,08,B0,03,4C,92,B0,20,D2,FF,A6
     ,58,E0,02,90,09,C6,59,20,D2, 2468     <049>
1019 DATA FF,C6,58,D0,F9,4C,8E,B0,48,4A,4A
     ,4A,4A,20,59,B1,68,29,0F,C9, 2419     <035>
1020 DATA 0A,90,02,69,06,69,30,4C,D2,FF,A2
     ,FC,9A,20,D1,B1,20,48,B2,20, 2261     <073>
1021 DATA EA,B1,20,9F,B2,A5,FC,20,4E,B1,A5
     ,FB,20,4E,B1,20,ED,B1,A9,3A, 2860     <148>
1022 DATA A0,20,20,F2,B1,A9,00,85,59,20,8E
     ,B0,20,ED,B1,A4,59,20,EF,B0, 2530     <233>
1023 DATA 91,FB,C8,84,59,C0,08,90,EC,20,10
     ,B2,A9,12,20,D2,FF,20,8E,B0, 2657     <105>
1024 DATA 20,EF,B0,C5,FF,F0,0D,20,43,B3,A9
     ,14,A0,14,20,F2,B1,4C,A2,B1, 2665     <034>
1025 DATA A9,92,20,D2,FF,20,33,B2,20,E0,B2
     ,20,3F,B2,90,9F,4C,8B,B5,A9, 2648     <123>
1026 DATA 93,20,D2,FF,A2,00,A9,03,9D,00,D8
     ,9D,00,D9,9D,00,DA,9D,00,DB, 2476     <237>
1027 DATA E8,D0,EF,60,A9,0D,2C,A9,20,4C,D2
     ,FF,20,D2,FF,98,4C,D2,FF,20, 2965     <160>
1028 DATA E4,FF,F0,FB,60,84,5D,85,5C,A0,00
     ,B1,5C,F0,06,20,D2,FF,C8,D0, 3100     <077>
1029 DATA F6,60,A5,FB,85,5A,A0,00,84,5B,B1
     ,FB,18,65,5A,85,5A,90,02,E6, 2606     <156>
1030 DATA 5B,06,5A,26,5B,C8,C0,08,90,EC,A5
     ,5A,65,5B,85,FF,60,18,A5,FB, 2467     <219>
1031 DATA 69,08,85,FB,90,02,E6,FC,60,A5,FB
     ,C5,5F,A5,FC,E5,60,60,A0,B3, 3106     <183>
1032 DATA A9,FB,20,FF,B1,A0,01,B9,00,02,20
     ,D2,FF,CC,00,02,C8,90,F4,A9, 2692     <098>
1033 DATA 10,ED,00,02,AA,20,ED,B1,CA,D0,FA
     ,A5,62,20,4E,B1,A5,61,20,4E, 2453     <236>
1034 DATA B1,20,ED,B1,A5,60,20,4E,B1,A5,5F
     ,20,4E,B1,A9,9F,20,D2,FF,20, 2575     <038>
1035 DATA EA,B1,24,5E,10,01,60,A9,12,20,D2
     ,FF,A2,28,20,ED,B1,CA,D0,FA, 2646     <161>
1036 DATA A9,92,4C,D2,FF,A5,D6,C9,16,B0,01
     ,60,A9,A0,85,A4,A9,78,85,A6, 2945     <204>
1037 DATA A9,04,85,A5,85,A7,A2,13,A0,27,B1
     ,A4,91,A6,88,10,F9,CA,F0,19, 2671     <208>
1038 DATA 18,A5,A4,69,28,85,A4,90,02,E6,A5
     ,18,A5,A6,69,28,85,A6,90,E0, 2503     <251>
1039 DATA E6,A7,4C,B6,B2,A9,91,4C,D2,FF,A9
     ,0F,8D,18,D4,A9,00,8D,05,D4, 2776     <000>
1040 DATA A9,F7,8D,06,D4,A9,11,8D,04,D4,A9
     ,32,8D,01,D4,A9,00,8D,00,D4, 2413     <126>
1041 DATA A0,80,20,09,B3,A9,10,8D,04,D4,60
     ,A2,FF,CA,D0,FD,88,D0,F8,60, 2914     <240>
1042 DATA A9,0F,8D,18,D4,A9,2D,8D,05,D4,A9
     ,A5,8D,06,D4,A9,21,8D,04,D4, 2385     <119>
1043 DATA A9,07,8D,01,D4,A9,05,8D,00,D4,A0
     ,FF,20,09,B3,A9,20,8D,04,D4, 2250     <078>
1044 DATA A9,00,8D,01,D4,8D,00,D4,60,38,20
     ,F0,FF,8A,48,98,48,18,A0,06, 2179     <175>
1045 DATA A2,18,20,F0,FF,A0,B4,A9,0A,20,FF
     ,B1,20,12,B3,20,E4,FF,F0,FB, 2931     <093>
1046 DATA A2,1D,A9,14,20,D2,FF,CA,D0,FA,68
     ,A8,68,AA,18,4C,F0,FF,0D,0D, 2704     <088>
1047 DATA 0D,20,20,20,20,20,20,20,4D,41,53
     ,43,48,49,4E,45,4E,53,50,52, 1144     <216>
1048 DATA 41,43,48,45,20,2D,20,45,44,49,54
     ,4F,52,20,0D,0D,20,20,20,20, 1023     <038>
1049 DATA 20,20,20,20,56,4F,4E,20,4E,2E,4D
     ,41,4E,4E,20,26,20,44,2E,57, 1128     <206>
1050 DATA 45,49,4E,45,43,4B,00,0D,0D,0D,20
     ,20,20,50,52,4F,47,52,41,4D, 1102     <117>
1051 DATA 4D,4E,41,4D,45,20,3A,20,00,0D,0D
     ,20,20,20,53,54,41,52,54,41, 1073     <095>
1052 DATA 44,52,45,53,53,45,20,3A,20,24,00
     ,0D,0D,20,20,20,45,4E,44,41, 1014     <129>
1053 DATA 44,52,45,53,53,45,20,20,20,3A,20
     ,24,00,92,05,20,50,52,4F,47, 1171     <217>
1054 DATA 52,41,4D,4D,20,3A,20,00,12,20,20
     ,2A,2A,2A,20,46,41,4C,53,43, 1024     <027>
1055 DATA 48,45,20,45,49,4E,47,41,42,45,20
     ,2A,2A,2A,20,20,92,00,0D,0D, 1058     <098>
1056 DATA 2A,2A,2A,20,45,4E,44,45,20,2A,2A
     ,2A,00,13,05,20,20,12,44,92, 920      <148>
1057 DATA 49,53,4B,20,4F,44,45,52,20,12,54
     ,92,41,50,45,0D,00,13,20,20, 1151     <035>
1058 DATA 49,2F,4F,20,2D,20,46,45,48,4C,45
     ,52,00,20,D1,B1,20,48,B2,A0, 1606     <012>
1059 DATA B3,A9,CF,20,FF,B1,20,8E,B4,85,FC
     ,20,8E,B4,85,FB,C5,61,A5,FC, 3207     <251>
1060 DATA E5,62,90,23,A5,FB,C5,5F,A5,FC,E5
     ,60,B0,19,20,A7,B4,D0,14,60, 2860     <112>
1061 DATA 20,A7,B4,F0,0C,85,F9,20,A7,B4,F0
     ,05,85,F8,4C,EF,B0,68,68,20, 2749     <088>
1062 DATA 43,B3,4C,5F,B4,20,CF,FF,C9,4C,D0
     ,09,20,D1,B1,20,48,B2,4C,0B, 2372     <046>
1063 DATA B6,C9,0D,60,A9,00,85,5E,20,5F,B4
     ,20,EA,B1,20,0D,B5,24,5E,30, 2042     <120>
1064 DATA 05,20,E4,FF,F0,FB,20,E1,FF,F0,26
     ,20,9F,B2,24,5E,10,09,20,4E, 2435     <198>
1065 DATA B5,20,0D,B5,20,60,B5,20,33,B2,20
     ,3F,B2,90,D7,A0,B4,A9,28,20, 2190     <207>
1066 DATA FF,B1,20,E4,FF,C9,0D,D0,F9,A9,00
     ,85,5E,A5,61,85,FB,A5,62,85, 3056     <240>
1067 DATA FC,20,E0,B2,4C,64,B1,A5,FC,20,4E
     ,B1,A5,FB,85,FF,20,4E,B1,A9, 3003     <221>
1068 DATA 20,A0,3A,20,F2,B1,A0,00,20,ED,B1
     ,B1,FB,20,4E,B1,C8,C0,08,90, 2566     <070>
1069 DATA F3,20,ED,B1,24,5E,30,03,A9,12,2C
     ,A9,20,20,D2,FF,20,10,B2,A5, 2190     <059>
1070 DATA FF,20,4E,B1,A9,92,20,D2,FF,4C,EA
     ,B1,A9,FF,85,B8,85,B9,A9,04, 3073     <029>
1071 DATA 85,BA,20,C0,FF,A2,FF,4C,C9,FF,20
     ,CC,FF,A9,FF,4C,C3,FF,20,5F, 3315     <189>
1072 DATA B4,A9,80,85,5E,20,4E,B5,20,48,B2
     ,A2,24,A9,2D,20,D2,FF,CA,D0, 2596     <111>
1073 DATA FA,20,EA,B1,20,EA,B1,20,60,B5,4C
     ,C1,B4,20,B8,B5,A6,5F,A4,60, 2812     <015>
1074 DATA A9,61,20,D8,FF,B0,0A,20,B7,FF,29
     ,BF,D0,03,4C,FB,B4,A9,01,20, 2577     <201>
1075 DATA C3,FF,20,68,B6,A0,B4,A9,4F,20,FF
     ,B1,20,F9,B1,4C,FB,B4,20,68, 2921     <237>
1076 DATA B6,A9,37,A0,B4,20,FF,B1,20,F9,B1
     ,A2,08,C9,44,F0,06,A2,01,C9, 2717     <213>
1077 DATA 54,D0,F1,A9,01,A8,20,BA,FF,A0,00
     ,E0,01,F0,1A,A9,40,8D,20,02, 2403     <101>
1078 DATA A9,3A,8D,21,02,B9,01,02,99,22,02
     ,C8,CC,00,02,90,F4,C8,C8,D0, 2182     <127>
1079 DATA 0C,B9,01,02,99,20,02,C8,CC,00,02
     ,D0,F4,98,A2,20,A0,02,4C,BD, 2018     <025>
1080 DATA FF,20,B8,B5,A5,BA,C9,08,90,33,A6
     ,B9,86,57,A9,01,20,C3,FF,A9, 2800     <022>
1081 DATA 60,85,B9,20,C0,FF,B0,28,A5,BA,20
     ,B4,FF,A5,B9,20,96,FF,20,A5, 2911     <053>
1082 DATA FF,85,61,A5,90,4A,4A,B0,13,20,A5
     ,FF,85,62,20,AB,FF,A5,57,85, 2663     <214>
1083 DATA B9,A9,00,20,D5,FF,90,03,4C,A3,B5
     ,86,5F,84,60,A5,BA,C9,01,D0, 2639     <131>
1084 DATA 0A,AD,3D,03,85,61,AD,3E,03,85,62
     ,4C,FB,B4,A9,13,20,D2,FF,A2, 2300     <120>
1085 DATA 1C,20,ED,B1,CA,D0,FA,60, 1230   <214>
```


MSE (Schluß). Dieses Listing können Sie (müssen aber nicht) mit dem Checksummer 64 V3 eingeben.

Butler – Eine Hilfe für Basic- und Assemblerfans

Machen Sie mehr aus Ihrem C128. »Butler« gestattet Ihnen ein komfortableres Arbeiten mit dem Computer. Der Editor wurde verbessert, und neben einer Reihe von nützlichen Befehlen lassen sich von nun an auch Maschinenroutinen des Z80-Prozessors vom Basic aus aufrufen.

Bei dem vorliegenden Programm BUTLER handelt es sich um eine Betriebssystemerweiterung, die zu Beginn aktiviert werden muß, dann aber stets verfügbar bleibt. Das Programm setzt sich aus zwei Teilen zusammen, von denen der eine im freien Speicherbereich von $1300 bis $1BFF und der andere am oberen Ende des Textspeichers zwischen $F000 und $FEFF liegt. Durch Änderung des Zeigers, der das Ende des verfügbaren Textspeichers definiert, schützt es sich vor Überschreiben durch einen Basic-Text.

Verwendungsmöglichkeiten

Das Programm ermöglicht ein komfortableres Arbeiten mit dem C128. Es ist nur im C128-Modus verwendbar, da es intensiv Betriebssystemroutinen nutzt und speziell auf die Besonderheiten und zahlreichen Möglichkeiten des C128 zugeschnitten ist.

Das Betriebssystem wurde im wesentlichen an zwei Stellen erweitert: Die Verwendung der Tastatur wurde erleichtert und um einige Funktionen erweitert. Das ohnehin umfangreiche Basic 7.0 wurde mit weiteren nützlichen Befehlen und Funktionen ausgestattet.

Aber nicht nur für Basic-Programmierer dürfte dieses Programm interessant sein, sondern auch Freunden der Maschinensprache bietet es viele Unterroutinen, auf die zurückgegriffen werden kann. Detailliertere Information ist dem betreffenden Abschnitt dieser Beschreibung zu entnehmen.

Aktivierung

Um das Programm zu starten, benutzt man am einfachsten den RUN-Befehl:

```
RUN "BUTLER"
```

Das kleine Basic-Programm lädt die Dateien »/BUTLER.1« und »/BUTLER.2« (Dieser Programmteil muß erst generiert werden. Lesen Sie dazu die Eingabehinweise am Schluß dieses Artikels.) mit dem BOOT-Befehl in den Speicher, wobei sie auch gleich initialisiert werden. Die Erweiterung bleibt aktiv, bis der RESET-Taster betätigt oder mittels dem Basic-Aufruf

```
SYS DEC("1303")
```

die Abschaltung veranlaßt wird. Eine erneute Aktivierung ist mit

```
SYS DEC("1300")
```

möglich.

Tastaturerweiterungen

Vereinfachte Eingabe von Escape-Sequenzen

Wer fand es nicht auch schon lästig, beim Einfügen oder Löschen von Bildschirmzeilen ständig wechselnd die <ESC>-Taste und den jeweiligen Buchstaben drücken zu müssen? All jenen ist geholfen! Nun kann man die <ALT>-Taste gedrückt halten und dann die betreffende Kommandotaste betätigen. Dies funktioniert bei allen Escape-Sequenzen und natürlich mit automatischer Wiederholung, falls die Kommandotaste ebenfalls gedrückt gehalten wird.

Cursorfunktionen

Nichts wesentlich Neues, aber zur Verwendung für den einen oder anderen vielleicht vorteilhafter, sind folgende neue Escape-Sequenzen:

<ESC> + <HOME> oder <ALT+HOME>: Cursor an den Anfang der untersten Fensterzeile

<ESC> + <Ü> oder <ALT+Ü>: Fenster ab der aktuellen Cursorposition löschen (für deutsche Tastatur)

<ESC> + <INS/DEL> oder <ALT+INS/DEL>: Fensterzeile löschen und restlichen Teil des Fensters nach oben schieben (=ESC+D)

<ESC> + <SHIFT+INS/DEL> oder <ALT+SHIFT+INS/DEL>: Fensterzeile einfügen; dazu alle Zeilen ab der aktuellen nach unten schieben <ESC> + <I>

LISTfunktionen

Neben den einfacheren Funktionen des letzten Abschnitts bietet der BUTLER aber auch komplexere Kommandos, um das gespeicherte Basic-Programm zu LISTen oder über den Bildschirm zu scrollen. Die Kommandos können durch Escape-Sequenzen oder als ALT-Tastenkombinationen eingegeben werden und stellen keine Basic-Befehle dar. Dabei ist die aktuelle Position des Cursors stets gleichgültig.

Um die Kommandos erklären zu können, ist etwas zu deren Arbeitsweise vorauszuschicken. Wird ein Kommando ausgelöst, dann wird das Bildschirmfenster nach einer Zeilennummer abgesucht, und zwar je nach Kommando entweder von oben nach unten, um die oberste oder von unten nach oben, um die unterste Zeilennummer zu finden. Damit die Suchzeit möglichst kurz bleibt, wird eine Zeilennummer nur erkannt, wenn sie in der ersten Spalte beginnt. Doch dies ist ja normalerweise immer der Fall. Wird keine Nummer gefunden, behandelt das Programm dies als sinnvollen Sonderfall.

Doch nun zu den Kommandos:

<ESC> + <CRSR>-unten beziehungsweise <ALT+CRSR>-unten: weiterblättern
unterste Zeilennummer suchen; falls keine existiert ab Programmanfang, sonst ab der Basic-Zeile mit nächstgrößerer Nummer so viele Zeilen listen, wie im Fenster Platz haben. Falls nicht genügend nachfolgende Basic-Zeilen existieren, wird stets die letzte Seite des Programms gelistet.

<ESC> + <CRSR>-hoch beziehungsweise <ALT+CRSR>-hoch: zurückblättern
oberste Zeilennummer suchen; falls keine existiert bis Programmende, sonst bis zur Basic-Zeile mit nächstkleinerer Nummer so viele Zeilen listen, wie im Fenster Platz haben. Falls nicht genügend vorausgehende Basic-Zeilen existieren, wird die erste Seite des Programms gelistet.

<ESC> + <CRSR>-rechts beziehungsweise <ALT+CRSR>-rechts: Zeile weiter

unterste Zeilennummer suchen; falls keine existiert, die erste Seite des Programms listen, sonst die Basic-Zeile mit nächstgrößerer Nummer unter der Zeile mit der gefundenen Zeilennummer ausgeben.
<ESC> + <CRSR>-links beziehungsweise <ALT+CRSR>-links: Zeile zurück
oberste Zeilennummer suchen; falls keine existiert, die letzte Seite des Programms listen, sonst die Basic-Zeile mit nächstkleinerer Nummer in der Fensterzeile vor der Zeile mit der oberen Zeilennummer ausgeben.
<ESC> + <LF> beziehungsweise <ALT+LF>: Seite noch einmal
oberste Zeilennummer suchen; falls keine existiert ab Programmanfang, sonst ab der zugehörigen Basic-Zeile so viele Zeilen ausgeben, wie in das Fenster passen. Falls nicht genügend Basic-Zeilen existieren, um das Fenster zu füllen, wird die letzte Seite des Programms gelistet.
<ALT+HELP>: Bereich um Fehlerzeile listen
wie der Basic-Befehl HELP, nur werden hier außer der fehlerhaften Zeile zusätzlich einige Zeilen davor und danach ausgegeben.

Wichtige Hinweise

Anmerkung: Beim Listen der letzten Programmseite oder beim Zurückblättern führen Basic-Zeilen, die sich über mehr als eine Fensterzeile erstrecken, dazu, daß einige Zeilen unnötig ausgegeben werden.

Bemerkungen zur Verwendung:

Da die Kommandos sich stets am aktuellen Fensterinhalt orientieren, ist es gleichgültig, wie bereits gelistete Zeilen auf den Bildschirm gekommen sind. Man kann sich also jederzeit mit dem LIST-Befehl einen Zeilenbereich ausgeben lassen und dann zeilenweise weitersrcollen oder seitenweise weiterblättern.

Möchte man sich das Basic-Programm von vorn seitenweise ansehen, löscht man einfach das Bildschirmfenster und blättert mit <ESC> + <CRSR>-unten oder <ALT+CRSR>-unten. Will man von hinten anfangen, löscht man ebenfalls das Fenster und blättert entsprechend mit <ESC> + <CRSR>-oben oder <ALT+CRSR>-oben.

Ist man sich über die Arbeitsweise der Kommandos im klaren, dann lassen sich diese noch vorteilhafter einsetzen, indem man eine Zeilennummer geeignet im Fenster positioniert und dann eines der Kommandos auslöst.

Basic-Erweiterungen

Speicherbefehle:

Wer in Basic mit Bytes hantiert, mußte bisher mit PEEK und POKE vorliebnehmen. Doch gerade bei so einfachen, maschinennahen Operationen macht sich die relativ langsame Arbeitsgeschwindigkeit schnell bemerkbar, weil man in der Regel nicht nur einzelne Bytes, sondern ganze Speicherbereiche zu bewältigen hat. Will man trotzdem nicht gleich auf Assembler ausweichen, sind die neuen Operationen dieses Abschnitts bestimmt nützlich.

Wie bei PEEK und POKE muß vor Ausführung eines Befehls die zu benutzende Speicherbank angewählt werden. Dies kann wie gewohnt mit dem BANK-Befehl erfolgen. Da es jedoch zum Beispiel beim Kopierbefehl auch möglich sein soll, von einer Speicherbank auf eine andere umzukopieren, sind manchmal zwei vorzuwählende Speicherbänke erforderlich. Diese können mit folgendem Befehl eingestellt werden:

```
BANKS  BankNr1,BankNr2
```

für bestimmte Operationen Speicherbank BankNr1 und BankNr2 vorwählen; der erste Operand überschreibt die mit dem BANK-Befehl ausgewählte Speicherbank; der zweite Operand wird gesondert gespeichert.

Im folgenden gilt für die Befehlsoperanden normalerweise stets die erste Banknummer. Nur bei Befehlen mit zwei Speicherbereichen wird für den zweiten Bereich die zweite Banknummer verwendet. Der Befehl BANKS wird in den nächsten Abschnitten noch mehrfach auftauchen.

Der Erklärung der Speicherbefehle ist jeweils ein Beispiel angefügt. Die Beispiele operieren im 40-Zeichen-Bildschirmspeicher, so daß die Wirkungsweise der Befehle direkt mitverfolgt werden kann.

Doch nun die Speicherbefehle im einzelnen:

```
MEMFILL  Adr1,Adr2,Wert
```

Speicherbereich von Adr1 bis einschließlich Adr2 mit Wert füllen. Beispiel:

```
BANK 0
FOR I=0 TO 127:MEMFILL 1024,2023,I:SLEEP 1:NEXT
MEMCOPY  Adr1a,Adr1b,Adr2a
```

Speicherbereich von Adr1a bis einschließlich Adr1b in den Bereich ab Adr2a kopieren. Beispiel:

```
BANKS 0,1:MEMCOPY 1024,2023,30000
SCNCLR 0
BANKS 1,0:MEMCOPY 30000,30999,1024
MEMSWAP  Adr1a,Adr1b,Adr2a
```

Speicherbereich von Adr1a bis einschließlich Adr1b mit dem Bereich ab Adr2a austauschen. Beispiel:

```
BANKS 0,1
FOR I=1 TO 10:MEMSWAP 1024,2023,30000:NEXT
MEMINV  Adr1,Adr2
```

Speicherbereich von Adr1 bis einschließlich Adr2 invertieren, also die Reihenfolge aller Bytes im angegebenen Bereich umdrehen. Beispiel:

```
BANK 0
FOR I=1 TO 10:MEMINV 1024,2023:SLEEP 1:NEXT
MEMROT  Adr1,Adr2,Anz
```

Speicherbereich von Adr1 bis einschließlich Adr2 gemäß Anz vorwärts rotieren lassen. Dies bewirkt, daß der Bereich von Adr1 bis Adr1+Anz ans Ende, der restliche Bereich zum Anfang kopiert wird. Beispiel:

```
BANK 0:MEMROT 1024,2023,40
```

Anmerkung: Der Befehl eignet sich nicht dazu, den 40-Zeichen-Bildschirm rotieren zu lassen, da die Daten zwischenzeitlich scheinbar wild durcheinander geraten. Der Befehl ist vielmehr bei der Vertauschung von aneinandergrenzenden Datenbereichen unterschiedlicher Größe nützlich, wenn kein Zwischenspeicher zur Verfügung steht.

```
MEMLOC (Adr1,Adr2,String)
```

Speicherbereich von Adr1 bis einschließlich Adr2 nach dem ersten Auftreten der Zeichenkette String absuchen. Eine Zeichenkette wird nur erkannt, wenn sie mit voller Länge im angegebenen Bereich liegt. Falls die Suche erfolgreich war, wird die Anfangsadresse, sonst stets Null zurückgegeben. Beispiel:

```
BANK 0:PRINT MEMLOC (1024,2023,"123")
```

Anmerkung: Möchte man den Bildschirm nach einer anderen Zeichenkette absuchen, ist der spezielle Bildschirmcode zu beachten. Bei Ziffern deckt er sich mit dem normalen Code.

Adreßoperationen

Möchte man auf zwei aufeinanderfolgende Byte zugreifen, weil diese beispielsweise einen Zeiger repräsentieren, so ist dies mit PEEK und POKE etwas umständlich. Ferner ist es im Basic oft gar nicht möglich, den Wert eines Sprungvektors zu verändern, der in der Unterbrechungsroutine verwendet

wird, ohne daß zwischenzeitlich ein inkonsistenter Sprungvektor entsteht, der dann zum Absturz des Computers führt.

Erleichterung schaffen folgende neuen Operationen:

```
CHG  Adr,Wert
```

Speicherstellen Adr und Adr+1 mit dem niederwertigen und dem höherwertigen Byte des Operanden Wert beschreiben. Unterbrechungen werden während des Änderns nicht zugelassen.

```
VEC  (Adr)
```

Werte der Speicherstellen Adr und Adr+1 auslesen und in eine Adresse umrechnen. Bei Adr muß das niederwertige, bei Adr+1 das höherwertige Byte stehen. Beispiel:

```
BANK 0
PRINT VEC(4624)-VEC(45)      Länge des Basic-Textes
CHG 4633,DEC("8000")         USR-Vektor ändern
PRINT USR(1)                 und verwenden
CHG 808,DEC("F4A2")          <RUN/STOP>-
                             Taste inaktivieren
CHG 808,DEC("F66E")          und wieder aktivieren
10 INPUT"NR";B               Nummer einlesen,
20 A=45:BANK 0               Anfangsadresse
30 DO:A=VEC(A):NR=VEC(A+2)   der zugehörigen
40 LOOP UNTIL A=0 OR NR=B    Basic-Zeile suchen
50 PRINT"ADR: ";A            und ausgeben
```

Mit den beiden letzten Operationen läßt sich also das Zerlegen eines Wertes in niederwertiges und höherwertiges Byte vermeiden. Sollten diese Bytes aber dennoch von Interesse sein, lassen sie sich mit folgenden zwei Funktionen leicht berechnen:

```
LOW  (Wert)     niederwertiges Byte von Wert
HIGH  (Wert)    höherwertiges Byte von Wert
```

Beispiel:

```
PRINT LOW(64000)
PRINT HIGH(64000)
```

Keiner langen Vorrede bedarf der Befehl

```
OLD
```

zum Wiederherstellen eines mit NEW gelöschten Basic-Programms. Häufiger wird diese Anweisung jedoch dann angewendet, wenn nach Drücken des RESET-Knopfs das Programm wiederhergestellt werden soll. Da bei der Neuinitialisierung nach Drücken des RESET-Knopfs auch die Basic-Erweiterung inaktiviert wird, ist es in diesem Fall zunächst erforderlich, sie mit SYS DEC("1300") erneut zu aktivieren.

Vielseitiger als der OLD-Befehl ist der Befehl

`HIDE  Nummer` oder `HIDE`,

der es ermöglicht, den Anfang des Basic-Textes vorübergehend zur angegebenen Zeile oder ganz ans Programmende zu legen, so daß der vordere Teil des Programms abgespalten bleibt, bis er mit dem Befehl

```
UNITE
```

wieder mit dem restlichen Programmtext verbunden wird.

Manipulieren und Editieren des Programmtextes

Die beiden letzten Befehle eröffnen interessante Anwendungen.

– schnelleres Editieren großer Programme:

Mit HIDE den vorderen Teil abtrennen. Dann läßt sich der Rest bequemer editieren, weil das Eingeben von Zeilen schneller erfolgt. Zum Schluß ist mit UNITE wieder alles zu verbinden.

– getrenntes Speichern eines hinteren Programmteils:

Mit HIDE an der gewünschten Stelle das Programm trennen, hinteren Teil wie gewohnt speichern, und dann entweder mit UNITE alles wieder verbinden oder aber mit NEW den gespeicherten Teil löschen und erst dann UNITE verwenden.

– Verbinden von Programmteilen:

Ersten Programmteil laden, mit HIDE verstecken, zweiten Programmteil normal laden, umnumerieren und dann mit UNITE anhängen.

– Programmteile während des Programmlaufs austauschen oder nachladen: Beispiel:

```
1 HIDE 10
2 BLOAD"ROUTINE 1",ON B0,P(VEC(45))
3 UNITE:GOSUB 9
4 HIDE 10
5 BLOAD"ROUTINE 2",ON B0,P(VEC(45))
6 UNITE:GOSUB 9
7 HIDE 10:BANK 0:CHG VEC(45),0:UNITE
8 END
9 REM *** Aufrufadresse ***
```

Dieses Beispiel setzt voraus, daß auf der Diskette die zwei Unterprogramme ROUTINE 1 und ROUTINE 2 stehen, die mit einer Zeilennummer beginnen. Ferner müssen die Routinen mit RETURN enden, da sie mit GOSUB aufgerufen werden.

Wie diesem Beispiel zu entnehmen ist, kann das Programm im versteckten Teil weiterlaufen. In diesem Teil werden jedoch keine Zeilennummern mehr gefunden, da der Basic-Interpreter mit der Suche erst beim restlichen Teil beginnt.

Nach so vielen Erläuterungen nun noch zu einem anderen Befehl, der weniger mit Programmzeigern, sondern viel mehr mit Programmzeilen hantiert:

```
TRANS Nummer1a,Nummer1b,Nummer2
```

Alle Zeilen des Zeilenbereichs von Nummer1a bis Nummer1b vor der Zeile mit Nummer2 einfügen.

Achtung: Die Nummern werden beim Kopieren nicht verändert. Daher sollten die Zeilen anschließend mit dem RENUMBER-Befehl neu durchnumeriert werden. Da das Neunumerieren nur dann durchgeführt wird, wenn alle Zeilenreferenzen aufgelöst werden können, sollte dies schon vorher sichergestellt sein, da die Korrektur des Programms nach dem Kopieren wegen der dann nicht streng steigenden Zeilennumerierung oft nicht mehr möglich ist.

Der letzte Befehl ist also zwar sehr nützlich, aber auch äußerst tückisch, wenn man nicht genügend aufpaßt. Vorsicht ist stets angebracht und nur geübte Benutzer sollten diesen Befehl bei ungesicherten Programmen anwenden.

Aufruf des Z80-Prozessors von Basic aus

In der Regel führt der Z80-Prozessor im C128 ein stilles Dasein. Wer mit der Programmierung dieses Mikroprozessors vertraut ist, kann ihn jetzt vom Basic aus mit dem Befehl

```
Z80CALL  Adresse
```

aufrufen. Die Unterroutine muß stets in der Speicherbank 0 stehen.

Die Verwendung dieses Prozessors kann mitunter sinnvoll sein, da er zum Beispiel einen mächtigen Kopierbefehl kennt.

Beispiele:

im Monitor einzugeben:

```
>0A000 01 00 20 11 00 80 21 00
>0A008 20 ED B0 C9
$01 $00 $20        LD BC,2000H    (Anzahl)
$11 $00 $80        LD DE,8000H    (Ziel)
$21 $00 $20        LD HL,2000H    (Quelle)
$ED $B0            LDIR
$C9                RET
```

Aufruf vom Basic aus:

```
Z80CALL DEC("A000")
```

Diese kurze Routine kopiert 8 KByte von Adresse $2000

bis $3FFF nach Adresse $8000 bis $9FFF, und dies einiges schneller als der 8502-Prozessor.
Folgendes ist wieder im Monitor einzugeben:

```
>0B000 01 FF 1F 11 01 20 21 00
>0B008 20 36 01 ED B0 C9
$01 $FF $1F        LD BC,1FFFH    (Anzahl)
$11 $01 $20        LD DE,2001H    (Ziel)
$21 $00 $20        LD HL,2000H    (Quelle)
$36 $01            LD (HL),1      (Wert)
$ED $B0            LDIR
$C9                RET
```

Aufruf vom Basic aus:

```
Z80CALL DEC("B000")
```

Bei dieser Routine wird ebenfalls der Blockkopierbefehl LDIR verwendet, hier jedoch dient er dazu, den Speicherbereich von $2000 bis $3FFF mit 1 zu füllen. Dies ist möglich, weil der Blockkopierbefehl von vorn nach hinten kopiert.

Anmerkung: Im Adreßbereich von $0000 bis $0FFF ist für den Z80 ein spezielles ROM eingeblendet, so daß er auf das RAM in diesem Adreßbereich leider nicht zugreifen kann.

Flexible Erweiterungen

Die bisher vorgestellten Erweiterungen sind alle möglichst allgemein gehalten und bilden in der Regel nur eine Basis dessen, was man sich vielleicht für seine persönlichen Problemstellungen wünschen würde. Um weitergehenden Wünschen entgegenzukommen, bietet BUTLER die Möglichkeit, die Tastatur- und vor allem die Basic-Erweiterungen einfach zu ergänzen oder umzudefinieren:

NEWESC Wert,Adresse

Neue Escape-Sequenz für das Zeichen mit ASCII-Code Wert definieren. Bei der Eingabe der zugehörigen Escape-Sequenz beziehungsweise ALT-Tastenkombination wird die Routine unter der angegebenen Adresse aufgerufen. Als Speicherbank wird die zuvor eingestellte verwendet. Beispiel:

```
BANK 15
NEWESC ASC("1"),DEC("CABC")
NEWESC ASC("2"),DEC("CACA")
```

<ESC> + <1> beziehungsweise <ALT+1> scrollt danach den Bildschirm hoch. <ESC> + <2> beziehungsweise <ALT+2> scrollt analog dazu abwärts.

NEWCMD Name,Adresse

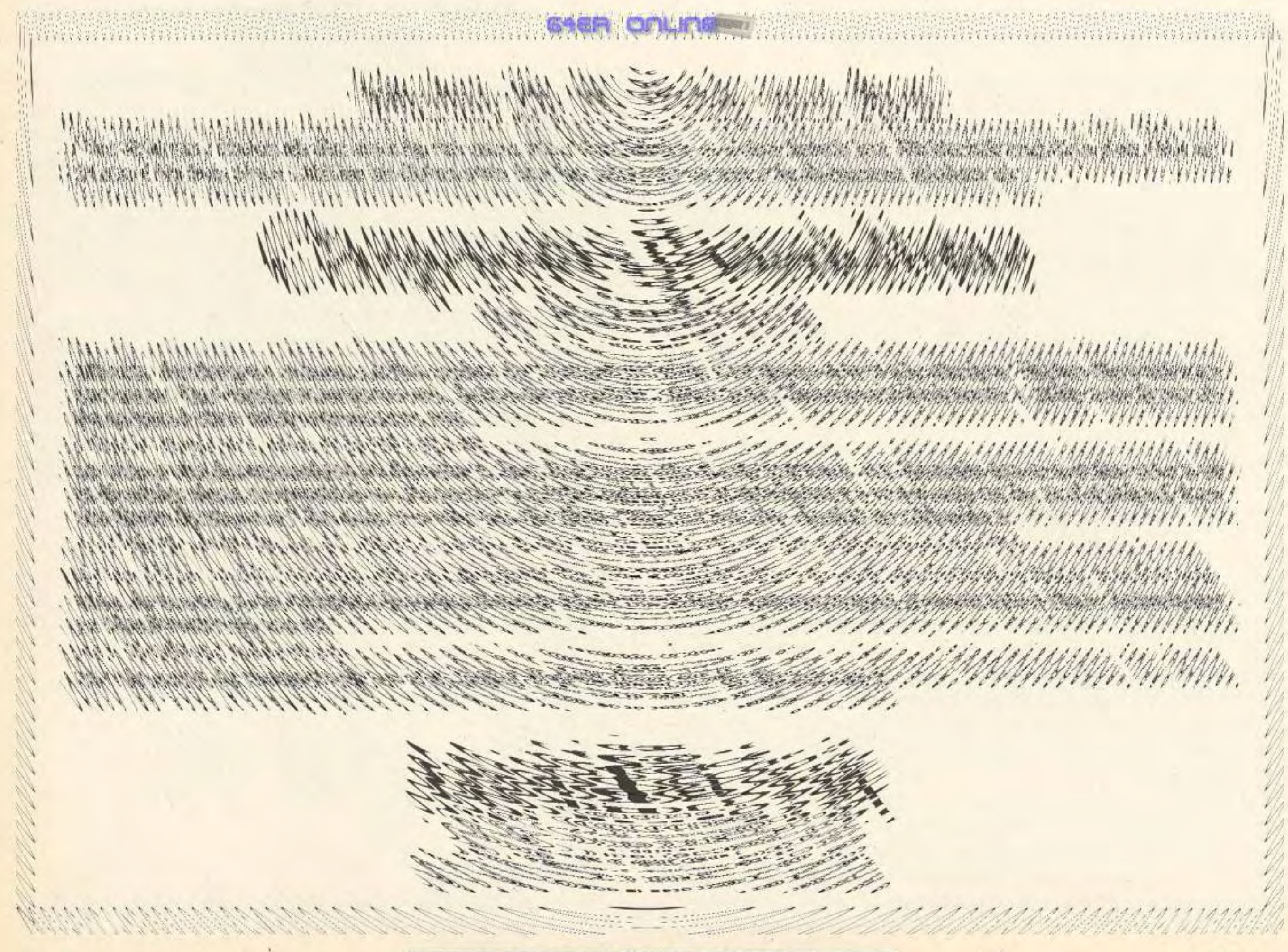
Neuen Basic-Befehl definieren. Die Routine ab der angegebenen Adresse auf der eingestellten Bank muß eventuelle Parameter selbst einlesen und auswerten. Beispiel:

```
BANK 15
NEWCMD "BELL",DEC("C98E")
BELL
```

NEWFN Name,Adresse

Neue Basic-Funktion vereinbaren. Ein Funktionsparameter wird automatisch ausgewertet, die restlichen müssen selbst eingelesen werden. Beispiel:

```
BANK 1
POKE 30000,96:REM Code für RTS
NEWFN "DUMMY",30000
PRINT DUMMY(99)
PRINT DUMMY("ABC")
```



Anmerkung: Wie an diesem Beispiel erkennbar ist, muß die Typprüfung von der Funktionsroutine vorgenommen werden.

So schön die Möglichkeiten der drei letzten Befehle auch sind, so setzen sie doch einige Erfahrung mit der Programmierung geeigneter Routinen voraus. Die notwendige Erfahrung kann man durch Analyse entsprechender Betriebssystemroutinen erlangen.

Codierung der neuen Befehle

Sicherlich von Interesse dürfte noch sein, wie die neuen Befehle verschlüsselt werden. Das Betriebssystem ermöglicht es, Erweiterungen durch zwei Byte ($CE $xx bei Funktionen, $FE $xx bei Anweisungen) zu kodieren. Das Betriebssystem selbst verwendet bereits die Kodierungen bis $CE $0A, beziehungsweise $FE $26. Die neuen Funktionen und Befehle von BUTLER verwenden Kodierungen bis $CE $0E, beziehungsweise $FE $37. Für künftige Erweiterungen sind auch die Kodierungen bis $CE $1A, beziehungsweise $FE $56 reserviert. Somit werden die selbstdefinierten Erweiterungen ab $CE $1B oder $FE $57 durchnumeriert. Dies geschieht in der Reihenfolge der Vereinbarungen und kann nicht beeinflußt werden. Mit dem Befehl

```
CLRNEWS
```

kann lediglich sichergestellt werden, daß die Kodezuweisung stets ab den angegebenen Werten beginnt. Dabei werden alle zuvor definierten Erweiterungen inaktiviert. Programme, die sich Erweiterungen selbst definieren, sollten diesen Befehl verwenden, um eine definierte Startumgebung vorzufinden.

Die Wirkung des letzten Befehls, aber viel mehr noch eine gesamte Neuinitialisierung von BUTLER und einiger wichtiger Betriebssystemdaten werden durch den Befehl

```
RESET
```

erreicht. Dieser Befehl ist besonders dann sinnvoll, wenn Sprungvektoren zwischenzeitlich verändert wurden und nun wieder ihre alten Werte erhalten sollen.

Schnittstelle zur Maschinenprogrammebene

Bankübergreifende Programmierung

Beim C 128 verfügt man zwar über verschiedene Speicherbänke, die vom Basic aus relativ einfach zu handhaben sind, doch sobald Maschinenprogramme zu schreiben sind, die auf eine andere Bank zugreifen sollen, erweist sich das Speicherbankkonzept als umständlich. Schon bei der Verwendung von Betriebssystemroutinen zeigen sich die ersten Probleme. Um eine Routine direkt aufrufen zu können, muß das betreffende ROM bereits beim Aufruf eingeblendet sein. Dies schränkt die Lage und die Größe des Maschinenprogramms ein. Ferner ist nicht stets gewährleistet, daß beim Rücksprung aus der Betriebssystemroutine das aufrufende Programmstück noch (oder wieder) sichtbar ist. Um letzterem Problem aus dem Weg zu gehen, aber auch um Routinen »unter« das Betriebssystem legen zu können, verfügt BUTLER über eine spezielle Aufrufroutine, mit der Unterprogramme im ROM bequem aufgerufen werden können. Ebenso existiert eine Routine, die es gestattet, ein Unterprogramm in einer beliebigen Speicherbank aufzurufen.

Die Verwendung sei an Beispielen erläutert:

```
JSR $1396    ;ROM-Aufruf veranlassen
.WORD Adr    ;ROM-Adresse low/high
JSR $13C7    ;Bank-Aufruf veranlassen
.BYTE Bank   ;Bank-Nr (0 bis 15)
.WORD Adr    ;Adresse der Routine (low/high)
```

Ungewöhnlich mag erscheinen, daß direkt hinter dem Aufrufbefehl Daten stehen. Diese Daten werden von den speziellen Aufrufroutinen gelesen und der Programmzähler korrigiert, so daß beim Rücksprung das Programm hinter den Daten fortgesetzt wird. Diese Übergabetechnik erlaubt es, alle Register normal verwenden zu können, setzt aber auch voraus, daß stets mit JSR aufgerufen wird, selbst wenn der nächste Befehl »nur« ein RTS ist.

Sprünge ins ROM oder auf eine andere Bank sind mit der letzten Routine nicht möglich. Ist dies doch einmal notwendig, dann kann man sich eventuell wie folgt helfen:

```
LDA #Adr-1   ;höherwertiges Byte
PHA          ;merken
LDA #Adr-1   ;niederwertiges Byte
PHA          ;merken
LDA #Konfig  ;gewünschte Konfiguration
JMP $02DF    ;umkonfigurieren und springen
```

Bei diesem Beispiel darf die anzugebende Konfiguration nicht mit der Banknummer verwechselt werden. Welche Konfiguration für welche Speicherbank zu wählen ist, kann einer Tabelle entnommen werden, wobei die Banknummer als Index dient. Das Original dieser Tabelle steht im ROM ab $F7F0, eine Kopie aber auch in BANK 0 ab $1310, wenn BUTLER verwendet wird.

Wer sich bereits mit der banküberg reifenden Programmierung beschäftigt hat, wird sicherlich die vorhandenen Routinen ab $02CD und $02E3 kennen, doch deren Verwendung ist bisweilen recht umständlich und einschränkend, weil beim Aufruf Teile des ROMs aktiviert sein müssen.

Einbinden von Erweiterungen

Nicht nur direkt vom Basic aus, sondern auch auf Maschinenspracheebene lassen sich Erweiterungen definieren. Wie dies möglich ist, soll an kommentierten Beispielen erläutert werden.

Tastaturerweiterungen

Die notwendigen Daten für die Erweiterungen müssen in einer Liste abgelegt sein. Jeder Listeneintrag muß nachfolgende Struktur aufweisen:

Konfiguration, Aufrufadresse-1, ASCII-Code

Listeneinträge sind lückenlos aneinanderzuhängen, und das Ende der Liste ist nicht zu kennzeichnen. Statt dessen ist beim Aufruf die Anzahl der Listenelemente (im Akkumulator) zu übergeben. Da diese Liste nicht notwendigerweise in der Speicherbank 0 stehen muß, ist auch die Banknummer (im X-Register) anzugeben. Schließlich muß noch die Anfangsadresse der Liste in der Zeropage bei $24/$25 bereitgestellt werden. Beispiel:

```
        LDA #Liste       ;Zeiger auf
        LDY #Liste       ;Liste
        STA $24          ;in der Zeropage
        STY $25          ;ablegen
        LDX #0           ;BankNr der Liste
        LDA #2           ;Elementanzahl
        JMP $F15D        ;Tastatur erweitern

Liste   .BYTE $3F        ;Konfiguration
        .WORD Beleg1-1   ;Adr-1 (low,high)
        .BYTE '1'        ;ESC 1

        .BYTE $3F        ;Konfiguration
        .WORD Beleg2-1   ;Adr-1 (low,high)
        .BYTE '2'        ;ESC 2

Beleg1  LDA #Tab1        ;Zeiger auf
        LDX #Tab1        ;erste Fkts-Tastenbelegung
        LDY #TabLg1      ;und deren Länge
        BNE DefBel
```

```
Beleg2 LDA #Tab2        ;Zeiger auf
       LDX #Tab2        ;zweite Fkts-Tastenbelegung
       LDY #TabLg2      ;und deren Länge
DefBel STA $24          ;Zeiger in
       STX $25          ;Zeropage ablegen
KopBel DEY              ;
       LDA ($24),Y      ;neue Funktions-
       STA $1000,Y      ;tastenbelegung
       TYA              ;umkopieren
       BNE KopBel       ;
       RTS

Tab1 .BLOCK 10,6,10,5,6,4,5,8,4,5           ;Stringlängen
     .BLOCK 'PRINT DS$',13,'DLOAD"'         ;F1,F2
     .BLOCK 'DIRECTORY',13,'BOOT"'          ;F3,F4
     .BLOCK 'DSAVE"',      'RUN"'           ;F5,F6
     .BLOCK 'LIST',13,     'MONITOR',13;F7,F8
     .BLOCK 'RUN',13,      'HELP',13        ;RUN,HELP
TabLg1 .EQU *-Tab1

Tab2 .BLOCK 7,5,6,7,4,5,3,6,4,5             ;Stringlängen
```

```
100 PRINT
110 PRINT "***{3SPACE}BUTLER{3SPACE}***
120 PRINT "(C)HJB{4SPACE}01.06.86
130 BANK 0
140 BOOT "/BUTLER.1"
150 BOOT "/BUTLER.2"
160 POKE 62442,169
```

Listing 1. »Butler«-Ladeprogramm

```
     .BLOCK 'GRAPHIC',   'COLOR'         ;F1,F2
     .BLOCK 'LOCATE',    'SCNCLR',13     ;F3,F4
     .BLOCK 'DRAW',      'PAINT'         ;F5,F6
     .BLOCK 'BOX',       'CIRCLE'        ;F7,F8
     .BLOCK 'RUN',13,    'HELP',13       ;RUN,HELP
TabLg2 .EQU *-Tab2
```

Basic-Erweiterungen

Ganz analog verläuft die Definition von Basic-Erweiterungen, nur die Struktur der Listenelemente ist etwas anders: Typ, BankNr, Aufrufadresse, Name, Null

Als Typen sind die Werte 0 (bei Funktionen) und 1 (bei Befehlen) zulässig. Der Name der Funktion beziehungsweise des Befehls ist direkt in die Liste einzutragen und muß durch eine Null abgeschlossen werden. Die Anfangsadresse der Liste muß in $5A/$5B bereitstehen. Beispiel:

```
      LDA #Liste        ;Zeiger auf
      LDY #Liste        ;Liste
      STA $5A           ;in der Zeropage
      STY $5B           ;ablegen
      LDX #0            ;BankNr der Liste
      LDA #2            ;Elementanzahl
      JMP $F23E         ;Basic erweitern

Liste .BYTE 0           ;Typ=Funktion
      .BYTE 0           ;BankNr
      .WORD LnAdr       ;Adr (low,high)
      .BLOCK 'LNADR'    ;Name ('line adress')
      .BYTE 0           ;Ende des Namens

      .BYTE 1           ;Typ=Befehl
      .BYTE 0           ;BankNr
```

Fortsetzung auf Seite 117


```
Name : /butler.1            1300 173e
-------------------------------------
1300 : 4c 06 13 4c 34 13 08 78   8b
1308 : 20 34 13 20 54 13 28 60   4a
1310 : 3f 7f bf ff 16 56 96 d6   1b
1318 : 2a 6a aa ea 06 0a 01 00   34
1320 : 08 48 a9 00 8d 00 ff 68   60
1328 : 28 60 08 48 a9 3f 8d 00   56
1330 : ff 68 28 60 08 78 20 20   7f
1338 : 13 20 51 42 a2 1f bd 73   f9
1340 : e0 9d 14 03 ca 10 f7 a2   a7
1348 : 09 bd 65 c0 9d 34 03 ca   be
1350 : 10 f7 28 60 20 b7 14 20   c2
1358 : c7 13 00 e2 02 20 c7 13   6c
1360 : 01 e2 02 60 00 28 43 29   00
1368 : 48 4a 42 01 06 56 d8 a9   50
1370 : 7f 8d 0d dd ac 0d dd 30   c0
1378 : 1a 20 3d f6 20 e1 ff d0   83
1380 : 12 20 56 e0 20 09 e1 20   66
1388 : 00 c0 20 06 13 20 20 13   8a
1390 : 4c 5c fa 4c 5f fa 08 48   d1
1398 : 8a 48 98 48 ba bd 05 01   25
13a0 : 85 ce bd 06 01 85 cf a0   79
13a8 : 02 fe 05 01 d0 03 fe 06   b8
13b0 : 01 b1 ce 99 e9 03 88 d0   eb
13b8 : f0 a9 01 8d ed 03 68 a8   59
13c0 : 68 aa a9 00 4c e4 03 08   f0
13c8 : 48 8a 48 98 48 ba bd 05   d6
13d0 : 01 85 ce bd 06 01 85 cf   1d
13d8 : a0 03 d0 03 99 e9 03 fe   81
13e0 : 05 01 d0 03 fe 06 01 b1   82
13e8 : ce 88 d0 f0 aa bd 10 13   4b
13f0 : 85 ce a9 01 8d ed 03 68   8c
13f8 : a8 68 aa a5 ce 4c e4 03   1c
1400 : 8d 00 ff 68 28 20 00 00   1e
1408 : 8d 01 ff 60 a2 0b bd 00   9b
1410 : 14 9d e4 03 ca 10 f7 60   5a
1418 : 8d aa 02 48 bd 10 13 8d   48
1420 : 27 14 68 60 86 79 a2 00   36
1428 : 20 a2 02 08 a6 79 28 60   b3
1430 : 8d b9 02 48 bd 10 13 8d   e7
1438 : 40 14 68 60 08 86 79 a2   88
1440 : 00 20 af 02 a6 79 28 60   14
1448 : 8d c8 02 48 bd 10 13 8d   87
1450 : 57 14 68 60 86 79 a2 00   96
1458 : 20 be 02 08 a6 79 28 60   f1
1460 : a0 ff 8d ea 03 8c eb 03   0b
1468 : a9 01 8d ed 03 60 20 7c   60
1470 : 14 8d 01 ff 60 20 7c 14   ac
1478 : 8d 02 ff 60 20 20 13 6c   3a
1480 : ea 03 20 89 14 8d 01 ff   d7
1488 : 60 8d 02 ff 6c ea 03 20   9a
1490 : 20 13 20 cf ff 8d 01 ff   ac
1498 : 60 20 20 13 20 d2 ff 8d   27
14a0 : 01 ff 60 20 20 13 20 e1   9c
14a8 : ff 8d 01 ff 60 20 20 13   5c
14b0 : 20 e4 ff 8d 01 ff 60 20   c6
14b8 : 20 13 20 7d 41 20 ee e0   ac
14c0 : 20 0c 14 20 e2 02 20 e2   74
14c8 : 02 a9 6e 8d 18 03 a9 13   53
14d0 : 8d 19 03 a9 fd 8d 3c 03   23
14d8 : a9 14 8d 3d 03 a9 42 8d   38
14e0 : 38 03 a9 15 8d 39 03 4c   ee
14e8 : b1 15 ed 14 04 00 9e ca   72
14f0 : bd 00 51 ca 14 00 3c ca   23
14f8 : 94 00 60 15 13 48 8a 29   37
1500 : 0f c9 08 68 b0 03 4c ad   b3
1508 : c6 a6 d0 e8 ec 20 0a b0   cc
1510 : 0b 9d 4a 03 a9 1b 9d 49   59
1518 : 03 e8 86 d0 4c c4 c6 aa   a6
1520 : ad ea 14 85 da ad eb 14   eb
1528 : 85 db ad ec 14 f0 11 0a   c5
1530 : 0a a8 8a 8d 01 ff 88 d1   b8
1538 : da f0 06 88 88 88 d0 f6   1b
1540 : 18 60 20 1f 15 b0 06 20   a4
1548 : 20 13 4c c1 c9 a9 02 48   c0
1550 : a9 dc 48 88 b1 da 48 88   af
1558 : b1 da 48 88 b1 da 4c df   7c
1560 : 02 46 ef a6 e4 86 eb 4c   21
1568 : b1 cb a4 e6 84 ec 20 7d   30
1570 : c7 20 8d 15 20 63 c3 4c   12
1578 : 4e ca a4 e6 84 ec 20 7d   5c
1580 : c7 20 b5 cb 20 99 15 20   a1
1588 : b5 cb 4c 4e ca e6 eb 20   d3
1590 : 74 cb b0 f9 c6 eb 4c 5c   0b
1598 : c1 a6 e5 e4 eb b0 03 4c   ab
15a0 : 70 c8 24 f8 10 05 a5 e4   26
15a8 : 4c 6f c3 20 7c c3 4c 85   c3
15b0 : cb a9 21 8d fc 02 a9 16   fd
15b8 : 8d fd 02 a2 05 bd c9 15   a8
15c0 : 9d 0c 03 ca 10 f7 4c 92   95
15c8 : 16 f8 15 db 15 46 16 0b   0d
15d0 : 28 89 89 16 16 8c 8c 16   06
15d8 : 16 75 59 c9 02 90 16 e8   07
15e0 : dd cf 15 b0 10 7d d9 15   7f
15e8 : 85 79 bc d5 15 bd d7 15   dc
15f0 : a6 79 4c 6a 51 4c 32 51   96
15f8 : f0 0d 48 ac d5 15 ad d7   83
1600 : 15 20 e2 43 68 b0 03 4c   f7
1608 : 26 43 ad cf 15 69 73 c5   2b
1610 : 0d a9 8a a2 ff b0 07 e8   5c
1618 : 69 27 38 ed cf 15 4c b0   19
1620 : 43 cd cf 15 b0 1d c9 02   ff
1628 : 90 19 e9 0b 20 60 16 68   4f
1630 : 68 a5 16 48 a5 17 48 20   6e
1638 : 5a 16 68 85 17 68 85 16   5f
1640 : 4c 56 79 4c 6c 79 c9 02   5d
1648 : 90 f9 cd d0 15 b0 f4 6d   e8
1650 : cf 15 e9 31 20 60 16 20   e8
1658 : 80 03 20 d0 02 4c cd 02   3a
1660 : 85 79 0a 65 79 a8 ad d1   08
1668 : 15 85 06 ad d3 15 85 07   81
1670 : a9 3f 8d 00 ff b1 06 85   cd
1678 : 02 c8 b1 06 85 04 c8 b1   0a
1680 : 06 85 03 a9 00 8d 00 ff   ab
1688 : 60 0f 06 13 52 45 53 45   7b
1690 : d4 00 4c 3d 17 00 00 a9   e4
1698 : 00 8d 9d 16 60 00 20 83   17
16a0 : 16 20 6a 15 8d 01 ff 60   a5
16a8 : 20 83 16 20 7a 15 8d 01   9c
16b0 : ff 60 20 83 16 20 58 cb   b3
16b8 : 8d 01 ff 60 20 83 16 20   88
16c0 : ee 16 8d 01 ff 60 20 83   c7
16c8 : 16 20 64 50 8d 01 ff 60   b3
16d0 : 20 20 13 20 23 51 8d 01   be
16d8 : ff 60 20 83 16 20 80 03   ea
16e0 : 8d 01 ff 60 20 83 16 20   b0
16e8 : 80 03 8d 01 ff 60 ad 9d   62
16f0 : 16 f0 03 4c 0a ef ce 9d   5f
16f8 : 16 a5 ec 85 e9 a5 eb 85   53
1700 : e8 98 48 8a 48 a9 0d 4c   36
1708 : 6e c2 08 48 8a 48 98 48   c0
1710 : ad 00 ff 48 a0 00 ba fe   b9
1718 : 06 01 d0 03 fe 07 01 bd   db
1720 : 06 01 85 ce bd 07 01 85   05
1728 : cf b1 ce f0 05 20 99 14   81
1730 : 90 e4 68 8d 00 ff 68 a8   f1
1738 : 68 aa 68 28 60 60 1c ce   2c
```

Listing 2. »/Butler.1«. Bitte geben Sie dieses Listing mit dem MSE im C64-Modus ein.

```
        .WORD ListLn   ;Adr (low,high)
        .BLOCK 'LISTLN';Name ('list line')
        .BYTE 0        ;Ende des Namens

LnAdr   JSR $F0CF      ;Real nach Integer wandeln
        JSR $16C6      ;zugehörige Zeile suchen
        LDY $61        ;Adresse ins Y-Reg. und
        LDA $62        ;Akku. (low/high Byte)
        BCS AdrOk      ;falls Zeile gefunden
        LDA #0         ;sonst Null
        TAY            ;als Adresse
AdrOk   JMP $F105      ;Y/A nach Real wandeln

ListLn  JSR $F0DB      ;Integer-Wert einlesen
        JSR $16C6      ;zugehörige Zeile suchen
        BCC Ende       ;falls nicht gefunden: Ende
        LDX $16        ;sonst Nummer
        LDA $17        ;laden und
        JSR $16D0      ;Zeile auflisten
Ende    RTS
```

Das angeführte Beispiel enthält eine Funktion zur Bestimmung der Anfangsadresse einer Basic-Zeile und einen Befehl zum Auflisten einer einzelnen Basic-Zeile. Hierbei werden mehrfach BUTLER-Routinen verwendet, die ihrerseits Routinen im ROM aufrufen. Diese ROM-Routinen könnte man auch direkt aufrufen, wenn man die speziellen Aufrufroutinen verwendet, die am Anfang des Kapitels vorgestellt wurden.

Tips zur Speicherbelegung

Die mit dem BANK-Befehl vorgewählte Speicherbank wird bekanntlich an der Adresse $03D5 abgelegt. Die mit dem neuen BANKS-Befehl vorwählbare zweite Speicherbank ist an der Adresse $03FD zu finden. Die Speicherstellen $03FE und $03FF sind für andere Anwendungen reserviert. Außerdem ist darauf zu achten, daß der Bereich von $03E4 bis $03EF intensiv verwendet wird und nicht anderweitig verwendet werden kann.

BUTLER selbst besteht aus zwei Teilen, die in BANK 0 in den Bereichen von $1300 bis $19FF und von $F000 bis $FCFF liegen. Ab $1A00 bis maximal $1BFF ist die Tabelle der Basic-Erweiterungen zu finden. Die zugehörige Sprungtabelle beginnt bei $FD00, gefolgt von der Tabelle für die neuen Escape-Sequenzen ab $FE00.

Abschließend bleibt noch anzuführen, daß temporär die RS232-Ein- und Ausgabepuffer ab $0C00 beziehungsweise $0D00 verwendet werden. (J.Blechinger/ah)

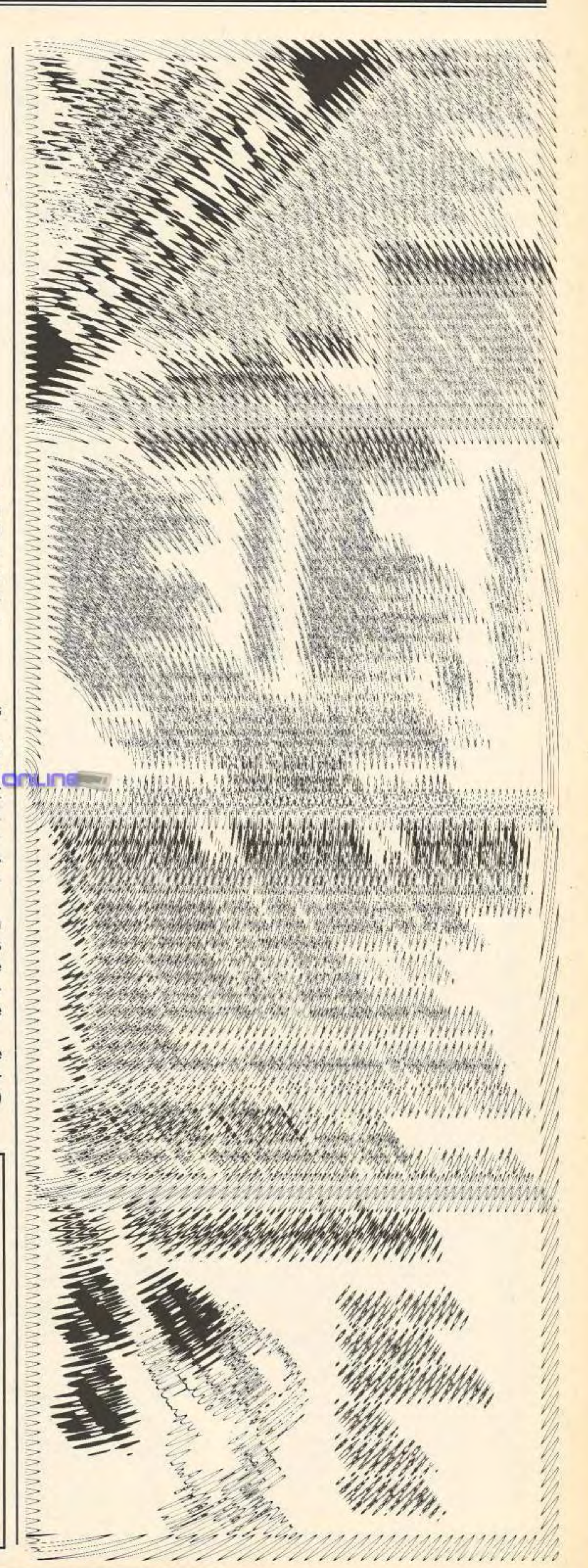
Eingabehinweise zum Programm »/BUTLER.$2000«

Das Programm ist mit dem MSE im C64-Modus einzugeben und zu speichern. Anschließend muß der C128 aus- und wieder eingeschaltet werden. Laden Sie nun das Programm mit LOAD"/BUTLER.$2000",8,1 im C128-Modus. Ist das geschehen, sind folgende Schritte erforderlich:

1. Mit dem Befehl MONITOR <RETURN> in den eingebauten Monitor springen.
2. Mit »T 2000 2CEB F000« das Programm »/BUTLER.$2000« nach $F000 schieben.
3. Mit »S"/BUTLER.2",08,F000,FCEB« das Programm speichern.

Wenn Sie die Leserservice-Diskette haben, entfallen diese Hinweise.

Von nun an kann das Programm mit »RUN "BUTLER"« geladen und gestartet werden.



```
Name : /butler.$2000          2000 2ceb
---------------------------------------
2000 : 20 2f f1 4c e2 02 48 e0   df
2008 : 00 d0 0a a9 b1 8d 1e f0   0a
2010 : 68 8d 1f f0 60 a9 2c 8d   44
2018 : 1e f0 68 4c 18 14 b1 00   3b
2020 : 60 4c 24 14 48 e0 00 d0   5f
2028 : 0a a9 91 8d 3c f0 68 8d   25
2030 : 3d f0 60 a9 2c 8d 3c f0   35
2038 : 68 4c 30 14 91 00 60 4c   88
2040 : 3c 14 a9 02 8d 50 f0 ba   c6
2048 : bd 01 01 bc 02 01 38 e9   3b
2050 : 02 b0 05 88 90 02 a0 00   98
2058 : 8d 69 f0 8c 71 f0 60 20   c8
2060 : 7b f0 08 48 ad 78 f0 18   f1
2068 : 69 00 8d 78 f0 ad 79 f0   88
2070 : 69 00 8d 79 f0 68 28 ad   ba
2078 : 77 f0 60 08 48 8a 48 98   ac
2080 : 48 ba bd 07 01 bc 08 01   8e
2088 : 85 ce 84 cf a0 03 fe 07   bb
2090 : 01 d0 03 fe 08 01 b1 ce   87
2098 : 99 76 f0 88 d0 f0 68 a8   41
20a0 : 68 aa 68 28 60 a9 2c 20   c1
20a8 : 96 13 5e 79 60 a2 0b 2c   2e
20b0 : a2 0e 2c a2 10 2c a2 16   d2
20b8 : 2c a2 17 2c a2 19 20 96   21
20c0 : 13 3c 4d 20 96 13 ef 77   f9
20c8 : 60 20 96 13 da 77 60 20   6b
20d0 : 96 13 15 88 60 20 96 13   ce
20d8 : f7 87 60 20 96 13 12 88   0a
20e0 : 60 20 96 13 0f 88 60 20   4f
20e8 : 96 13 03 88 60 20 96 13   61
20f0 : 09 88 60 20 96 13 1e 9e   11
20f8 : 60 20 96 13 f4 87 60 20   be
2100 : 96 13 d4 84 60 20 96 13   6d
2108 : c9 84 60 20 96 13 dd 77   98
2110 : 60 20 96 13 7b 87 60 20   3e
2118 : 96 13 7e 87 60 20 96 13   50
2120 : 6e 86 60 20 96 13 90 86   3f
2128 : 60 20 96 13 e3 86 60 a9   e8
2130 : 00 a0 f0 8d 5b 13 8c 5c   a7
2138 : 13 8d 12 12 8c 13 12 4c   1b
2140 : ae f1 a9 00 a0 fe 8d ea   5f
2148 : 14 8c eb 14 a9 00 8d ec   ca
2150 : 14 a9 04 a0 ed a2 14 84   9b
2158 : 24 86 25 a2 00 85 5c a9   4e
2160 : 24 20 18 14 20 77 f1 a5   ee
2168 : 24 18 69 04 85 24 90 02   33
2170 : e6 25 c6 5c d0 ee 60 a0   6d
2178 : 03 20 24 14 20 1f 15 a2   ac
2180 : 03 b0 1b ac ec 14 c0 04   2b
2188 : b0 18 ee ec 14 c8 98 0a   9b
2190 : 0a a8 20 a5 f1 20 24 14   84
2198 : 20 a5 f1 88 91 da ca 10   53
21a0 : f1 60 4c b3 f0 48 8a 48   57
21a8 : 98 aa 68 a8 68 60 20 42   53
21b0 : f1 4c 3c f4 00 ff fe 1b   a7
21b8 : 00 ff fe 1b 03 00 00 00   0b
21c0 : 00 00 01 0f 06 13 52 45   af
21c8 : 53 45 54 00 01 00 f8 f1   ab
21d0 : 43 4c 52 4e 45 57 53 00   f4
21d8 : 01 00 9c f2 4e 45 57 43   52
21e0 : 4d 44 00 01 00 9f f2 4e   d5
21e8 : 45 57 46 4e 00 01 00 75   27
21f0 : f2 4e 45 57 45 53 43 00   42
21f8 : 20 42 f1 a9 0b 8d cf 15   71
2200 : a9 27 8d d0 15 ad d1 15   ea
2208 : ac d3 15 8d d2 15 8c d4   47
2210 : 15 8d b4 f1 8c b6 f1 ad   f9
2218 : d5 15 ac d7 15 8d d6 15   e1
2220 : 8c d8 15 8d b5 f1 8c b7   9c
2228 : f1 85 24 84 25 a9 00 a8   66
2230 : 91 24 a9 05 a0 c2 a2 f1   6d
2238 : 84 5a 86 5b a2 00 85 5c   ef
2240 : a9 5a 20 18 14 a0 00 20   a8
2248 : 24 14 99 be f1 c8 c0 04   25
2250 : 90 f5 a2 00 20 24 14 f0   d9
2258 : 07 9d 00 0c e8 c8 d0 f4   b1
2260 : 8e bd f1 98 38 65 5a 85   7f
2268 : 5a 90 02 e6 5b 20 2e f3   bf
2270 : c6 5c d0 d1 60 20 f9 f0   a3
2278 : 8e c1 f1 20 e1 f0 aa 98   e9
2280 : d0 01 ca 8e bc bf f1 8e   40
2288 : c0 f1 ae d5 03 bd 10 13   2c
2290 : 8d be f1 a9 01 a0 be a2   83
2298 : f1 4c 57 f1 a2 01 2c a2   eb
22a0 : 00 8e be f1 20 11 f1 a8   79
22a8 : f0 19 c9 11 b0 18 20 c9   99
22b0 : f2 20 e1 f0 ae d5 03 8e   0b
22b8 : bf f1 8c c0 f1 8d c1 f1   21
22c0 : 4c 2e f3 4c ad f0 4c b9   b1
22c8 : f0 a8 8c bd f1 a9 24 a2   29
22d0 : 01 20 18 14 88 20 24 14   ac
22d8 : 99 00 0c 98 d0 f6 60 a9   21
22e0 : 00 99 00 0c ad d5 15 85   17
22e8 : 24 ad d7 15 85 25 a0 00   7f
22f0 : b1 24 18 f0 19 a5 3d 48   1c
22f8 : a5 3e 48 a9 00 85 3d a9   78
2300 : 0c 85 3e 20 96 13 e6 43   87
2308 : 68 85 3e 68 85 3d 60 84   9c
2310 : 79 a5 26 d0 02 c6 27 c6   80
2318 : 26 a5 26 c5 24 a5 27 e5   2b
2320 : 25 90 0a a0 00 b1 26 a4   93
2328 : 79 91 26 b0 e4 60 ad be   8f
2330 : f1 c9 02 b0 4a a0 bf f1   91
2338 : c9 11 b0 43 ac bd f1 c0   20
2340 : 11 b0 83 20 df f2 90 3a   da
2348 : c8 cc bd f1 d0 31 a5 0d   6b
2350 : e9 75 cd cf 15 a9 00 2a   54
2358 : cd be f1 d0 22 a5 0d 29   f1
2360 : 7f 06 0d 18 65 0d a8 ad   e5
2368 : d1 15 85 24 ad d3 15 85   83
2370 : 25 a2 00 bd bf f1 91 24   b8
2378 : e8 c8 e0 03 90 f5 60 4c   30
2380 : bc f0 a2 01 bd b4 f1 bc   40
2388 : b6 f1 18 7d bc f1 90 01   8c
2390 : c8 48 dd b8 f1 98 48 fd   0c
2398 : ba f1 90 03 4c b3 f0 ca   8b
23a0 : 10 e2 ae b5 f1 ac b7 f1   cb
23a8 : ad be f1 d0 13 ae d6 15   77
23b0 : 8a ac d8 15 18 6d bd f1   31
23b8 : 90 03 ee d8 15 8d d6 15   e4
23c0 : 86 24 84 25 ae b5 f1 ac   d8
23c8 : b7 f1 e8 d0 01 c8 86 26   89
23d0 : 84 27 ac bd f1 20 0f f3   0f
23d8 : 38 88 b9 00 0c 90 03 09   26
23e0 : 80 18 91 24 98 d0 f2 09   43
23e8 : 80 88 91 24 ae b4 f1 ac   47
23f0 : b6 f1 ad be f1 d0 12 ae   2d
23f8 : d2 15 8a ac d4 15 18 69   b6
2400 : 03 90 03 ee d4 15 8d d2   bc
2408 : 15 86 24 84 25 ad b4 f1   70
2410 : ac b6 f1 85 26 84 27 a0   a9
2418 : 03 20 0f f3 88 b9 bf f1   a7
2420 : 91 24 98 d0 f7 68 8d b6   6a
2428 : f1 68 8d b4 f1 68 8d b7   4f
2430 : f1 68 8d b5 f1 ae be f1   e3
2438 : fe cf 15 60 a9 00 a0 fd   88
2440 : 8d d1 15 8c d3 15 a9 00   19
2448 : a0 1a 8d d5 15 8c d7 15   53
2450 : a9 00 a0 fe 8d b8 f1 8c   81
2458 : ba f1 a9 ff a0 1b 8d b9   02
2460 : f1 8c bb f1 20 fb f1 a0   af
2468 : 73 a2 f4 a9 0b 20 38 f2   17
2470 : 4c 47 f8 00 00 f0 f4 4c   92
2478 : 4f 57 00 00 00 [illegible] [illegible] 48   6f
2480 : 49 47 48 00 00 00 27 f5   08
2488 : 56 45 43 00 01 00 fe f4   48
2490 : 43 48 47 00 01 00 15 f6   1c
2498 : 42 41 4e 4b 53 00 01 00   b1
24a0 : 28 f6 4d 45 4d 46 49 4c   04
24a8 : 4c 00 01 00 50 f6 4d 45   b1
24b0 : 4d 43 4f 50 59 00 01 00   16
24b8 : 81 f6 4d 45 4d 53 57 41   00
24c0 : 50 00 01 00 f2 f6 4d 45   f7
24c8 : 4d 49 4e 56 00 01 00 2a   75
24d0 : f7 4d 45 4d 52 4f 54 00   5a
24d8 : 00 00 ab f7 4d 45 4d 4c   8f
24e0 : 4f 43 00 00 00 00 00 00   d1
24e8 : 00 00 20 c9 f0 4c cf f0   bc
24f0 : 18 b0 38 08 20 ea f4 28   ed
24f8 : 90 01 a8 4c ff f0 20 db   7c
2500 : f0 a5 16 48 a5 17 48 20   c6
2508 : e1 f0 68 85 25 68 85 24   20
2510 : a9 24 ae d5 03 20 30 14   4c
2518 : 08 78 a0 01 b9 16 00 20   31
2520 : 3c 14 88 10 f7 28 60 20   0d
2528 : ea f4 a9 16 ae d5 03 20   9f
2530 : 18 14 a0 01 20 24 14 aa   63
2538 : 88 20 24 14 a8 8a 4c 05   76
2540 : f1 a5 5e 18 65 60 85 5e   cb
2548 : 90 02 e6 5f 60 a5 5e 38   9c
2550 : e5 60 85 5e b0 02 c6 5f   87
2558 : 60 a5 5a 38 e5 5c a8 85   17
2560 : 60 a5 5b e5 5d aa 05 60   26
2568 : f0 0d b0 0c a5 5a 38 e5   66
2570 : 60 85 5a b0 02 c6 5b 18   33
2578 : 60 a5 5c 38 e5 5a a8 85   a8
2580 : 60 a5 5d e5 5b aa e8 90   97
2588 : 0c a5 5c 38 e5 60 85 5c   b5
2590 : b0 02 c6 5d 38 60 a9 5a   80
2598 : 20 06 f0 98 aa a9 5e 20   bc
25a0 : 24 f0 20 59 f5 b0 1c 20   05
25a8 : 4d f5 8a f0 10 20 1e f0   0d
25b0 : 20 3c f0 c8 d0 f7 e6 5b   63
25b8 : e6 5f e8 d0 f0 20 1e f0   0c
25c0 : 20 3c f0 60 a9 5c 20 06   50
25c8 : f0 98 aa a9 5e 20 24 f0   3d
25d0 : 20 79 f5 90 18 20 4d f5   e0
25d8 : 4c df f5 c6 5d c6 5f 20   34
25e0 : 1e f0 20 3c f0 88 c0 ff   5c
25e8 : d0 f5 ca d0 ee 60 20 db   aa
25f0 : f0 84 5a 85 5b 20 e1 f0   8a
25f8 : 84 5c 85 5d 20 e1 f0 84   95
2600 : 5e 85 5f 60 20 db f0 84   b2
2608 : 5a 85 5b 20 e1 f0 84 5c   70
2610 : 85 5d 4c ed f0 20 96 13   a5
2618 : c9 6b 20 ed f0 e0 10 b0   14
2620 : 04 8e fd 03 60 4c b0 f0   58
2628 : 20 04 f6 8a ae d5 03 85   0a
2630 : 79 a9 5a 20 24 f0 20 59   15
2638 : f5 b0 14 a5 79 e0 00 f0   c0
2640 : 0b 20 3c f0 c8 d0 fa e6   55
2648 : 5b e8 d0 f5 20 3c f0 60   73
2650 : 20 ee f5 c4 5a e5 5b b0   a1
2658 : 09 ae d5 03 ac fd 03 4c   ed
2660 : 96 f5 a5 5c 38 e5 5a aa   57
2668 : a5 5d e5 5b a8 8a 18 65   ab
2670 : 5e 85 5e 98 65 5f 85 5f   62
2678 : ae d5 03 ac fd 03 4c c4   1a
2680 : f5 20 ee f5 ae d5 03 bd   21
2688 : 10 13 ae fd 03 bc 10 13   0a
2690 : 85 24 84 25 20 59 f5 b0   f3
2698 : 1a 20 4d f5 8a f0 11 8a   5e
26a0 : 48 20 b4 f6 c8 d0 fa e6   d1
26a8 : 5b e6 5f 68 aa e8 d0 ef   70
26b0 : 20 b4 f6 60 a2 3f e4 24   f4
26b8 : d0 0f e4 25 d0 0b b1 5a   ce
26c0 : aa b1 5e 91 5a 8a 91 5e   0a
26c8 : 60 a9 5a 8d aa 02 8d b9   a9
26d0 : 02 a6 24 20 a2 02 48 a9   e1
26d8 : 5e 8d aa 02 a6 25 20 a2   41
26e0 : 02 a6 24 20 af 02 a9 5e   b1
26e8 : 8d b9 02 68 a6 25 20 af   53
26f0 : 02 60 20 db f0 84 5a 85   4d
26f8 : 5b 20 e1 f0 84 5e 85 5f   0a
2700 : ae d5 03 bd 10 13 85 24   09
2708 : 85 25 a0 00 f0 11 20 b4   c9
2710 : f6 e6 5a d0 02 e6 5b a5   3a
2718 : 5e d0 02 c6 5f c6 5e a5   29
2720 : 5a c5 5e a5 5b e5 5f 90   2d
2728 : e5 60 20 db f0 8c e4 f4   b2
2730 : 8d e5 f4 20 e1 f0 8c e8   9b
2738 : f4 8d e9 f4 20 e1 f0 aa   36
2740 : 98 18 6d e4 f4 8d e6 f4   1d
2748 : 8a 6d e5 f4 8d e7 f4 ae   ea
2750 : d5 03 bd 10 13 85 24 85   11
2758 : 25 ac e8 f4 ad e9 f4 84   b3
2760 : 5e 85 5f cc e6 f4 ed e7   90
2768 : f4 90 3f ad e6 f4 ac e7   c2
2770 : f4 85 5a 84 5b 20 0a f7   1d
2778 : ad e4 f4 ac e5 f4 85 5a   3b
2780 : 84 5b ae ef f4 ac e6 f4   95
2788 : d0 01 ca 88 84 5e 86 5f   b1
2790 : 20 0a f7 ad e4 f4 ac e5   dd
2798 : f4 85 5a 84 5b ad e8 f4   27
27a0 : ac e9 f4 85 5e 84 5f 20   f6
27a8 : 0a f7 60 20 ea f4 84 5a   e7
27b0 : 85 5b 20 e1 f0 84 5c 85   d7
27b8 : 5d 20 a5 f0 20 c3 f0 20   d1
27c0 : 1d f1 20 c9 f2 20 de f7   b3
27c8 : 20 13 f8 a0 00 98 90 0b   e1
27d0 : a5 5a 38 ed bd f1 a8 a5   c7
27d8 : 5b e9 00 4c 05 f1 a2 01   1e
27e0 : a0 00 98 9d 00 0d ec bd   f2
27e8 : f1 b0 27 98 f0 08 bd ff   54
27f0 : 0b d9 ff 0b d0 15 e8 c8   34
27f8 : bd ff 0b d9 ff 0b d0 04   56
2800 : b9 00 0d 24 98 9d 00 0d   12
2808 : 4c e6 f7 b9 00 0d a8 4c   a0
2810 : e6 f7 60 ae d5 03 a9 5a   b1
2818 : 20 06 f0 a2 00 a0 00 ec   ab
2820 : bd f1 e8 b0 21 a5 5c c5   62
2828 : 5a a5 5d e5 5b 90 17 20   40
2830 : 1e f0 dd ff 0b f0 06 bd   09
2838 : 00 0d aa d0 f2 e6 5a d0   f5
2840 : de e6 5b d0 da 18 60 a9   c5
2848 : 06 a0 53 a2 f8 20 57 f1   99
2850 : 4c 43 fb 3f f1 fa 1d 3f   0e
2858 : 1e fb 9d 3f 65 fa 11 3f   b4
2860 : 7a fa 91 3f a7 fa 0a 3f   9d
2868 : 85 fa 84 00 00 8c 6b f8   8f
2870 : 8c 6c f8 60 20 78 f8 8a   3b
2878 : ac 6b f8 99 00 0d c8 8c   f0
2880 : 6b f8 cc 6c f8 d0 18 ee   7c
2888 : 6c f8 38 60 20 90 f8 aa   4a
2890 : ac 6b f8 cc 6c f8 f0 f2   02
2898 : 88 8c 6b f8 b9 00 0d 18   60
28a0 : 60 20 96 13 50 c1 20 e6   7a
28a8 : f8 90 0c a5 eb c5 e4 b0   82
28b0 : 0a 20 9e 16 4c a6 f8 a5   5e
28b8 : 16 a4 17 60 20 96 13 61   b8
28c0 : 15 20 e6 f8 90 f1 a5 e5   b9
28c8 : c5 eb b0 ef 20 a8 16 4c   e5
28d0 : c1 f8 20 bc 16 4c db f8   d2
28d8 : 20 b2 16 c9 3a b0 06 38   c2
28e0 : e9 30 38 e9 d0 60 20 d8   6f
28e8 : f8 b0 3e 20 97 16 20 d2   1c
28f0 : f8 b0 36 a2 00 9d 00 02   13
28f8 : e8 e0 06 b0 2c 20 d2 f8   e9
2900 : 90 f3 e9 0d 8d 2a f9 a9   0b
2908 : 00 9d 00 02 a2 ff a0 01   c6
2910 : a5 3d 48 a5 3e 48 86 3d   d5
```




```
2918 : 84 3e 20 da 16 20 96 13  02
2920 : a0 50 68 85 3e 68 85 3d  6b
2928 : 18 60 00 00 00 20 6d f8  19
2930 : a5 2d a6 2e 85 61 86 62  1d
2938 : 20 74 f8 a0 01 b1 61 f0  e9
2940 : 1e a0 03 b1 61 c5 17 90  67
2948 : 09 d0 14 88 b1 61 c5 16  39
2950 : b0 0b a0 01 b1 61 aa 88  b0
2958 : b1 61 4c 34 f9 f0 01 18  af
2960 : 60 e8 8e 2b f9 20 2d f9  87
2968 : a9 00 8d 2c f9 2a 49 01  12
2970 : 48 d0 0a ad 2a f9 d0 05  18
2978 : ce 2b f9 f0 14 ce 2c f9  d5
2980 : 20 8c f8 b0 0c 85 61 86  ba
2988 : 62 ee 2c f9 ce 2b f9 d0  7b
2990 : ef ae 2c f9 68 60 a9 00  51
2998 : 8d 2c f9 8e 2b f9 20 c6  1c
29a0 : 16 08 ae 2b f9 f0 29 90  b8
29a8 : 06 ad 2a f9 f0 01 e8 a9  5d
29b0 : 00 8d 2c f9 ca f0 19 8e  77
29b8 : 2b f9 a0 01 b1 61 aa f0  db
29c0 : 0f 88 b1 61 85 61 86 62  ee
29c8 : ee 2c f9 ce 2b f9 d0 ea  c0
29d0 : ae 2c f9 28 a0 01 b1 61  b3
29d8 : 60 8d 0e fa 4c f1 f9 20  5e
29e0 : 15 fa 20 9e 16 a0 00 b1  18
29e8 : 61 aa c8 b1 61 86 61 85  e1
29f0 : 62 a0 01 b1 61 f0 54 20  48
29f8 : a3 14 f0 4f ad 0e fa f0  e4
2a00 : 06 ce 0e fa 4c df f9 a5  47
2a08 : eb c5 e4 90 d2 60 00 a0  92
2a10 : 01 b1 61 f0 36 a0 02 b1  34
2a18 : 61 aa c8 b1 61 20 d0 16  bd
2a20 : 20 96 13 76 ca a5 ec c5  38
2a28 : e6 f0 11 a5 eb 48 20 96  2e
2a30 : 13 8d 15 68 c5 eb 85 eb  06
2a38 : b0 0f e6 eb 20 96 13 5c  63
2a40 : c1 20 96 13 55 ca 20 a8  97
2a48 : 16 a9 01 60 85 16 84 17  c8
2a50 : 4c c6 16 48 a5 e4 38 e5  bc
2a58 : e5 aa 68 60 48 a5 e4 38  6e
2a60 : e5 e5 4a aa 68 60 20 bc  a3
2a68 : f8 b0 30 20 53 fa 20 96  83
2a70 : f9 f0 4f a2 01 20 96 f9  69
2a78 : 4c b6 fa 20 a1 f8 b0 42  0b
```

```
2a80 : 20 53 fa 4c e2 fa ae 08  63
2a88 : 12 e8 f0 0e ad 09 12 ac  d1
2a90 : 0a 12 20 4c fa 90 03 20  b5
2a98 : d7 fa 60 38 6e 2a f9 a9  7f
2aa0 : 00 a8 20 4c fa 4c b6 fa  69
2aa8 : 20 a1 f8 b0 ee 20 c6 16  24
2ab0 : a0 01 b1 61 f0 0c 20 96  86
2ab8 : 13 42 c1 46 f0 a9 00 4c  1a
2ac0 : d9 f9 38 6e 2a f9 20 53  0b
2ac8 : fa a9 ff a8 85 16 84 17  f5
2ad0 : 4c e2 fa 85 16 84 17 20  1f
2ad8 : 5c fa 20 96 f9 f0 e3 20  83
2ae0 : 5c fa 20 61 f9 8a 48 20  43
2ae8 : 96 13 42 c1 46 f0 68 4c  f7
2af0 : d9 f9 46 f0 20 bc f8 b0  a3
2af8 : a2 20 9e 16 a2 01 20 96  f5
2b00 : f9 d0 0b 8d 2a f9 a2 01  d5
2b08 : 20 61 f9 4c 13 fb ad 2a  fd
2b10 : f9 d0 03 20 a8 16 20 0f  10
2b18 : fa f0 03 20 9e 16 60 46  f8
2b20 : f0 20 a1 f8 b0 9c 20 a8  69
2b28 : 16 a2 01 20 61 f9 d0 09  0f
2b30 : ad 2a f9 f0 04 90 05 b0  c9
2b38 : 09 20 9e 16 20 0f fa 20  62
2b40 : a8 16 60 a0 52 a2 fb a9  9d
2b48 : 05 20 38 f2 20 83 fc 4c  74
2b50 : db fc 01 00 83 fb 4f 4c  d8
2b58 : 44 00 01 00 93 fb 48 49  a9
2b60 : 44 45 00 01 00 ac fb 55  67
2b68 : 4e 49 54 45 00 01 00 c4  aa
2b70 : fb 54 52 41 4e 53 00 01  d4
2b78 : 00 63 fc 5a 38 30 43 41  49
2b80 : 4c 4c 00 a9 01 a8 91 2d  1e
2b88 : 20 96 13 4f 4f 20 96 13  19
2b90 : 82 4f 60 d0 08 a9 ff 85  c5
2b98 : 16 85 17 d0 03 20 db f0  d3
2ba0 : 20 c6 16 a5 61 a4 62 85  2d
2ba8 : 2d 84 2e 60 a9 1c a0 01  af
2bb0 : a6 76 f0 03 18 69 24 84  94
2bb8 : 2d 85 2e 4c 88 fb 20 96  d3
2bc0 : 13 f0 84 60 20 be fb 20  a1
2bc8 : 47 fc 8c e4 f4 8d e5 f4  8a
2bd0 : 20 a5 f0 20 47 fc 20 52  84
2bd8 : fc a4 61 a5 62 8c e6 f4  43
2be0 : 8d e7 f4 cc e4 f4 ed e5  b1
```

```
2be8 : f4 90 59 20 a5 f0 20 47  70
2bf0 : fc 8c e8 f4 8d e9 f4 cc  a1
2bf8 : e6 f4 ed e7 f4 b0 32 ac  c8
2c00 : e4 f4 ad e5 f4 cc e8 f4  ca
2c08 : ed e9 f4 90 37 ae e8 f4  af
2c10 : ac e9 f4 ad e6 f4 8d e8  c2
2c18 : f4 ad e7 f4 8d e9 f4 ad  d3
2c20 : e4 f4 8d e6 f4 ad e5 f4  fd
2c28 : 8d e7 f4 8e e4 f4 8c e5  ac
2c30 : f4 ad e8 f4 d0 03 ce e9  08
2c38 : f4 ce e8 f4 a2 00 20 52  bb
2c40 : f7 4c 88 fb 4c b0 f0 20  4d
2c48 : db f0 20 c6 16 a4 61 a5  d4
2c50 : 62 60 90 0e a0 01 b1 61  64
2c58 : aa f0 07 88 b1 61 85 61  4c
2c60 : 86 62 60 20 db f0 8c 9c  e4
2c68 : fc 8d 9d fc 08 78 ad 00  2d
2c70 : ff 48 a9 3e 8d 00 ff a9  f2
2c78 : b0 8d 05 d5 ea 68 8d 00  13
2c80 : ff 28 60 a9 c3 8d ee ff  45
2c88 : a9 93 8d ef ff a9 fc 8d  b8
2c90 : f0 ff 60 31 d0 ff 3e 3f  43
2c98 : 32 00 ff cd 00 00 c3 e0  55
2ca0 : ff a2 01 8e ca fc a9 30  9e
2ca8 : 8d cf fc a0 ca a2 fc a9  79
2cb0 : 01 20 38 f2 ee cf fc ae  ec
2cb8 : ca fc bd cf 15 dd c8 fc  c7
2cc0 : 90 e9 ce ca fc f0 e4 60  fd
2cc8 : 1b 57 00 0f e2 02 40 40  31
2cd0 : 40 00 28 43 29 48 4a 42  05
2cd8 : 01 06 56 20 c7 13 00 e2  51
2ce0 : 02 20 c7 13 01 e2 02 20  b6
2ce8 : a1 fc 60 49 2f 4f 20 44  bf
```

Listing 3. »/Butler.$2000« Auch dieses Listing ist mit dem MSE im C 64-Modus einzugeben. Beachten Sie bitte die Eingabehinweise im Text.

Tornado-Copy 1571

Man nehme: den großen Speicher des C128, beschleunige die ohnehin hohe Geschwindigkeit der Floppy 1571/ 1570 und kopiere so eine Diskettenseite in nur 75 Sekunden.

Sicherlich haben Sie sich als Besitzer eines C 128 schon des öfteren darüber geärgert, daß die bisherigen Kopierprogramme für den C 64 mindestens drei Durchgänge pro Diskettenseite benötigen. Da diese Programme aber im C 128-Modus nicht lauffähig sind, kann der doppelt große Speicher des C 128 nicht genutzt werden. Das DOS-Shell auf der Test-/Demodiskette nutzt zwar den gesamten Speicher, ist aber viel zu langsam. Tornado-Copy 1571 kopiert hingegen eine Diskettenseite in weniger als 75 Sekunden und in nur zwei Durchgängen. Diskettenseiten, auf denen mehr als 184 Blöcke frei sind, werden bei BAM-Auswahl sogar in nur einem Durchgang kopiert.

Geben Sie Tornado-Copy (Listing 1) bitte im C 64-Modus mit dem MSE ein und speichern das Programm auf Diskette. Zum Kopieren laden Sie das Programm im 128er-Modus mit DLOAD und starten es mit RUN. Tornado-Copy schaltet dann als erstes auf den FAST-Modus und den 80-Zeichen-Bildschirm um. Auf dem Bildschirm erscheinen nun eine Reihe von Fragen, die Sie zunächst beantworten müssen. Sie können wählen zwischen einem Kopiervorgang mit BAM-Auswahl, bei dem nur die belegten Blöcke kopiert werden, und einem Kopieren aller Diskettenblöcke einer Seite. Weiterhin können Sie die Zieldiskette wahlweise vor dem Beschreiben formatieren lassen.

(Ulrich Textoris/nj)

```
Name : tornado copy          1c01 2646
--------------------------------------
1c01 : 12 1c 0a 00 de 35 3a fe  22
1c09 : 25 3a 9e 30 39 37 36 30  80
1c11 : 00 00 00 a0 00 2e 1c 20  48
1c19 : b6 15 20 19 16 78 78 20  cc
1c21 : c2 1b 20 0b 17 78 20 24  d8
1c29 : 18 a9 0a 20 d6 19 20 db  0b
1c31 : 18 a5 6b c9 4a f0 e0 6c  b8
1c39 : fc ff 78 a9 ff 85 74 ad  e1
1c41 : 00 1c 49 08 8d 00 1c 20  2c
1c49 : 4e 06 29 7f 8d 01 02 0a  d2
1c51 : aa bd 1b 06 85 52 bd 1c  7c
1c59 : 06 85 53 20 25 06 20 31  60
1c61 : 06 4c 35 04 a2 10 bd 86  0a
1c69 : 04 9d 00 0f ca 10 f7 ad  86
1c71 : 01 0c 85 12 ad 02 0c 85  42
```

```
1c79 : 13 a9 24 8d ac 02 a9 01  9f
1c81 : 85 0c 85 0d a9 00 8d b2  46
1c89 : 01 85 5e 85 ff 4c 00 0f  16
1c91 : a9 f0 85 3b 85 03 58 a5  98
1c99 : 03 30 fc 78 6c fc ff a2  f6
1ca1 : 04 86 e7 a2 02 bd 00 03  4b
1ca9 : 95 9b ca 10 f8 ad 01 04  c9
1cb1 : 85 06 ad 02 04 85 07 a9  c1
1cb9 : 4c 8d 00 03 a9 13 8d 01  98
1cc1 : 03 a9 05 8d 02 03 a6 06  6b
1cc9 : e4 74 f0 03 a9 e0 2c a9  29
1cd1 : d0 85 00 58 a5 00 30 fc  84
1cd9 : c9 02 b0 0b a6 06 86 74  ce
1ce1 : 09 80 8d 00 04 78 60 a9  67
1ce9 : ff 85 74 c6 e7 f0 ed a5  aa
1cf1 : e7 c9 02 f0 22 a9 a1 8d  6d
1cf9 : 01 03 a9 05 8d 02 03 a9  cf
```

```
1d01 : 4c 8d 00 03 a9 e0 85 00  2c
1d09 : a5 00 30 fc a5 16 85 12  9f
1d11 : a5 17 85 13 4c a3 04 20  38
1d19 : af 05 4c a3 04 20 a4 05  b0
1d21 : ad 00 1c 29 10 d0 05 a9  ea
1d29 : 08 4c b5 99 a9 00 a0 3f  93
1d31 : 59 00 03 59 40 03 59 80  f9
1d39 : 03 59 c0 03 88 10 f1 85  55
1d41 : 3a 20 8f f7 20 0f 97 a0  88
1d49 : 09 2c 0f 18 30 fb 2c 00  c3
1d51 : 1c 88 d0 f5 a9 ff 8d 03  7b
1d59 : 1c ad 0c 1c 29 1f 09 c0  04
1d61 : 8d 0c 1c a9 ff a0 05 8d  65
1d69 : 01 1c 2c 0f 18 30 fb 2c  b1
1d71 : 00 1c 88 d0 f5 a0 bb b9  82
1d79 : 00 01 2c 0f 18 30 fb 8d  f5
1d81 : 01 1c c8 d0 f2 b1 30 2c  b2
```

```
1d89 : 0f 18 30 fb 8d 01 1c c8  13
1d91 : d0 f3 2c 0f 18 30 fb ad  96
1d99 : 0c 1c 09 e0 8d 0c 1c a9  0f
1da1 : 00 8d 03 1c 85 50 a9 01  30
1da9 : 4c b5 99 4c 4f 94 a2 02  e8
1db1 : b5 9b 9d 00 03 ca 10 f8  54
1db9 : 60 a9 01 85 06 a9 c0 85  9b
1dc1 : 00 58 a2 ff 86 74 e8 86  53
1dc9 : 07 a5 00 30 fc a5 4a d0  71
1dd1 : f8 78 09 80 8d 00 04 60  01
1dd9 : a2 04 86 e7 ad 01 04 85  1a
1de1 : 06 ad 02 04 85 07 a9 80  f7
1de9 : a6 74 e4 06 d0 02 09 08  15
1df1 : 85 00 58 a5 00 30 fc 78  a8
1df9 : 29 7f c9 02 b0 08 8d 00  16
1e01 : 04 a6 06 86 74 60 c6 e7  e0
1e09 : f0 f4 a9 ff 85 74 a5 e7  40
1e11 : c9 02 f0 0b a5 16 85 12  be
1e19 : a5 17 85 13 4c d2 05 20  bd
1e21 : af 05 4c d2 05 a0 ea ce  5f
1e29 : 05 af 05 95 04 5a 04 6c  f6
1e31 : 52 00 8d 00 04 8e 01 04  a7
1e39 : 8c 02 04 a2 fd a9 03 8d  70
1e41 : 7d 06 20 6b 06 a9 00 8d  00
1e49 : 7d 06 ad 00 04 30 05 a2  50
1e51 : 00 20 6b 06 ad 00 04 60  a9
1e59 : a2 fd a9 03 8d b5 06 20  a3
1e61 : 9f 06 a9 00 8d b5 06 ad  68
1e69 : 00 04 30 05 a2 00 20 9f  02
1e71 : 06 ad 00 04 60 20 95 06  38
1e79 : ad 00 18 cd 00 18 d0 f8  dc
1e81 : 45 37 29 04 f0 f2 bd 00  ca
1e89 : 03 8d 0c 40 a5 37 49 04  9f
1e91 : 85 37 a9 08 2c 0d 40 f0  2b
1e99 : fb e8 d0 dc 4c b2 81 20  79
1ea1 : ce 81 a5 37 29 fb 85 37  77
1ea9 : 60 2c 0d 40 ad 00 18 49  39
1eb1 : 08 8d 00 18 a9 08 2c 0d  29
1eb9 : 40 f0 fb ad 0c 40 9d 00  5f
1ec1 : 03 e8 d0 e8 ad 00 18 29  17
1ec9 : f7 8d 00 18 60 a9 23 85  75
1ed1 : fa a9 13 85 fb a9 30 8d  fe
1ed9 : 16 16 a9 04 8d 17 16 a9  22
1ee1 : 08 20 b1 ff a9 6f 20 93  23
1ee9 : ff a2 00 bd 13 16 20 a8  a5
1ef1 : ff e8 e0 06 d0 f5 a0 00  9c
1ef9 : b1 fa 20 a8 ff c8 c0 19  c0
1f01 : d0 f6 20 ae ff a5 fa 18  73
1f09 : 69 19 85 fa 90 02 e6 fb  6c
1f11 : 18 ad 16 16 69 19 8d 16  0a
1f19 : 16 90 02 ee 17 16 38 a9  2c
1f21 : b7 e5 fa a9 15 e5 fb b0  90
1f29 : b6 60 4d 2d 57 00 00 19  b0
1f31 : a9 08 20 b1 ff a9 6f 20  68
1f39 : 93 ff a0 00 b9 33 16 20  c2
1f41 : a8 ff c8 c0 05 d0 f5 4c  7a
1f49 : ae ff 4d 2d 45 30 04 2c  2e
1f51 : 0d dc ad 00 dd 09 10 8d  b9
1f59 : 00 dd a9 08 2c 0d dc f0  34
1f61 : f9 ad 00 dd 49 10 8d 00  38
1f69 : dd ad 0c dc 9d 00 0b e8  93
1f71 : d0 e8 ad 00 dd 29 ef 8d  23
1f79 : 00 dd 60 a9 40 85 fa 20  11
1f81 : 93 16 ad 00 dd cd 00 dd  93
1f89 : d0 f8 45 fa 29 40 f0 f2  c4
1f91 : bd 00 0b 8d 0c dc a5 fa  f7
1f99 : 49 40 85 fa a9 08 2c 0d  69
1fa1 : dc f0 fb e8 d0 dc 20 b8  f7
1fa9 : 16 60 ad 05 d5 09 08 8d  dc
1fb1 : 05 d5 a9 7f 8d 0d dc a9  03
1fb9 : 00 8d 05 dc a9 03 8d 04  4e
1fc1 : dc ad 0e dc 29 80 09 55  f8
1fc9 : 8d 0e dc 2c 0d dc 60 ad  af
1fd1 : 0e dc 29 80 09 08 8d 0e  cb
1fd9 : dc ad 05 d5 29 f7 8d 05  1a
1fe1 : d5 2c 0d dc 60 8d 00 0c  36
1fe9 : 8e 01 0c 8c 02 0c a2 fd  94
1ff1 : a9 03 8d 7a 16 20 64 16  ef
1ff9 : a9 00 8d 7a 16 ad 00 0c  3c
2001 : 30 05 a2 00 20 64 16 60  9b
2009 : a2 fd a9 03 8d 56 16 20  99
2011 : 38 16 a9 00 8d 56 16 ad  fe
2019 : 00 0c 30 05 a2 00 20 38  e7
2021 : 16 60 a5 70 f0 0c a9 81  f8
2029 : a2 12 a0 00 20 ce 16 20  0e
2031 : f1 16 a0 01 84 c8 a9 20  eb
2039 : 85 cb a9 04 85 ca 20 db  76
2041 : 17 20 9a 17 a9 00 85 c9  36
2049 : a5 70 f0 05 20 00 18 b0  c7
2051 : 0d a4 c9 a5 cb 91 fb a5  5c
2059 : ca 91 fd 20 7f 17 e6 c9  4f
2061 : a5 c9 a6 c8 dd 5b 17 90  e4
2069 : df e6 c8 a5 c8 c9 24 90  2f
2071 : d0 60 00 15 15 15 15 15  8d
2079 : 15 15 15 15 15 15 15 15  79
2081 : 15 15 15 15 13 13 13 13  45
2089 : 13 13 13 12 12 12 12 12  4b
2091 : 12 11 11 11 11 11 a6 cb  5e
2099 : e8 e0 ff 90 11 a5 ca 49  ff
20a1 : 80 85 ca 30 07 a2 20 e6  70
20a9 : ca 4c 97 17 a2 04 86 cb  5e
20b1 : 60 a9 00 85 fc 85 fe a5  da
20b9 : c8 0a 0a 26 fc 0a 26 fc  80
20c1 : 85 fd a6 fc 86 fe 0a 26  63
20c9 : fc 0a 26 fc 38 e5 fd 85  a9
20d1 : fb a5 fc e5 fe 85 fc a5  f6
20d9 : fb 18 69 e8 85 fd a5 fc  31
20e1 : 69 03 85 fe 18 a5 fb 69  7e
20e9 : ec 85 fb a5 fc 69 0b 85  9e
20f1 : fc 60 a2 04 86 fb a2 0f  37
20f9 : 86 fc a2 00 86 fd a2 07  97
2101 : 86 fe a0 50 a2 04 a9 00  29
2109 : 91 fb 91 fd 88 d0 f9 c6  40
2111 : fc c6 fe ca d0 f2 60 a5  fb
2119 : c8 0a 0a a8 c8 a2 00 b9  93
2121 : 00 0b 9d fd 03 c8 e8 e0  aa
2129 : 03 d0 f4 a6 c9 e8 4e ff  c3
2131 : 03 6e fe 03 6e fd 03 ca  04
2139 : d0 f4 60 a9 00 85 67 a9  ee
2141 : 01 85 c8 a9 00 85 c9 20  00
2149 : 9a 17 a9 00 85 63 20 ca  63
2151 : 19 20 ea 18 90 03 4c d6  38
2159 : 18 a5 67 f0 08 a9 00 20  4a
2161 : d6 19 20 db 18 a9 81 a6  69
2169 : c8 a4 c9 20 ce 16 20 f1  fc
2171 : 16 20 ea 18 b0 1c a9 81  eb
2179 : a6 c8 a4 c9 20 ce 16 20  f7
2181 : ab 19 20 f1 16 ad 00 0c  e6
2189 : c9 02 90 e5 20 33 1b 4c  d5
2191 : 5a 18 20 ab 19 a5 67 d0  73
2199 : 51 a9 12 85 c8 20 9a 17  1a
21a1 : a0 00 b1 fd 85 ca b1 fb  db
21a9 : 85 cb 20 b6 1b 20 97 1b  3a
21b1 : a2 0f a5 6d f0 09 bd 00  40
21b9 : 11 9d 90 0b ca 10 f7 a5  77
21c1 : 6c c9 01 f0 0d ad a2 0b  4f
21c9 : 85 6e ad a3 0b 85 6f 4c  98
21d1 : c5 18 a5 6e 8d a2 0b a5  3f
21d9 : 6f 8d a3 0b 20 ca 19 a5  61
21e1 : ca 85 68 a5 cb 85 6a 20  0f
21e9 : ab 19 4c c9 1a a9 80 4c  f7
21f1 : ce 16 58 20 cf ff [illegible] 6b  ce
21f9 : a9 0d 78 20 d2 ff 4c d2  4f
2201 : ff a5 c9 85 65 a5 c8 85  a8
2209 : 64 a5 ca 85 68 a5 cb 85  91
2211 : 6a a5 c9 a2 1e 86 66 18  f4
2219 : a4 63 d0 05 69 04 4c 0f  4a
2221 : 19 a4 c8 79 73 19 a4 c8  12
2229 : d9 5b 17 90 04 38 f9 5b  28
2231 : 17 a8 b1 fd 29 03 c5 67  59
2239 : d0 24 b1 fd a6 63 d0 0f  2e
2241 : 29 0c c9 04 d0 18 b1 fd  f4
2249 : 09 08 91 fd 4c 67 19 29  31
2251 : 1c c9 0c d0 09 b1 fd 09  97
2259 : 10 91 fd 4c 67 19 c6 66  62
2261 : f0 05 c8 98 4c 0f 19 e6  88
2269 : c8 a6 c8 e0 24 90 02 38  12
2271 : 60 20 9a 17 a5 65 38 e9  a5
2279 : 04 85 c9 4c fa 18 18 84  16
2281 : c9 b1 fb 85 cb b1 fd 85  20
2289 : ca 60 00 06 06 06 06 06  f9
2291 : 06 06 06 06 06 06 06 06  91
2299 : 06 06 06 06 05 05 05 05  7b
22a1 : 05 05 05 05 05 05 05 05  a1
22a9 : 05 05 05 05 05 05 05 8d  bb
22b1 : 01 ff bd 00 0b 8d 03 ff  4a
22b9 : 9d 00 80 e8 d0 f1 8e 00  6a
22c1 : ff 60 ad 03 d5 48 a5 68  c3
22c9 : 30 03 a9 3f 2c a9 7f 8d  f6
22d1 : 03 d5 a5 6a 8d ef 03 a2  1f
22d9 : 00 20 e4 03 68 8d 03 d5  2d
22e1 : 60 a2 14 bd 98 19 9d e4  e2
22e9 : 03 ca 10 f7 60 0a aa bd  d1
22f1 : f3 19 8d e7 19 bd f4 19  57
22f9 : 8d e8 19 a2 00 bd 00 80  84
2301 : f0 07 20 d2 ff e8 4c e6  1d
2309 : 19 60 09 1a 1d 1a 2f 1a  6c
2311 : 41 1a 4a 1a 54 1a 75 1a  55
2319 : 86 1a 8e 1a 96 1a a7 1a  a0
2321 : 0e d1 55 45 4c 4c 44 49  e1
2329 : 53 4b 20 45 49 4e 4c 45  95
2331 : 47 45 4e 00 d2 45 41 44  93
2339 : 2f d7 52 49 54 45 2d c5  c1
2341 : 52 52 4f 52 3a 00 da 49  7c
2349 : 45 4c 44 49 53 4b 20 45  89
2351 : 49 4e 4c 45 47 45 4e 00  55
2359 : aa 20 d4 52 41 43 4b 3a  63
2361 : 00 aa 20 d3 45 43 54 4f  97
2369 : 52 3a 00 93 0e 20 37 35  74
2371 : 53 20 2d 20 c3 cf d0 d9  d5
2379 : 20 02 28 42 59 20 d5 4c  73
2381 : 52 49 43 48 20 d4 2e 29  05
2389 : 11 11 0d 00 c6 4f 52 4d  31
2391 : 41 54 49 45 52 45 4e 3f  fe
2399 : 20 20 4e 9d 00 0d 20 20  3a
23a1 : c9 c4 3a 20 00 0d ce 41  85
23a9 : 4d 45 3a 20 00 c2 c1 cd  e4
23b1 : 2d c1 55 53 57 41 48 4c  b8
23b9 : 3f 20 20 4e 9d 00 0d ce  86
23c1 : 4f 43 48 20 45 49 4e 45  2a
23c9 : 20 c4 49 53 4b 45 54 54  e1
23d1 : 45 20 4b 4f 50 49 45 52  ec
23d9 : 45 4e 3f 20 20 4a 9d 00  e4
23e1 : 20 b6 1b a9 02 20 d6 19  07
23e9 : 20 db 18 a5 67 d0 14 a5  4a
23f1 : 6c f0 10 a9 84 a6 6e a4  8f
23f9 : 6f 20 ce 16 20 b6 15 20  3b
2401 : 19 16 78 a9 00 85 c9 a9  1f
2409 : 01 85 c8 85 63 20 9a 17  7f
2411 : 20 ea 18 20 97 1b 20 ea  59
2419 : 18 b0 1e a9 03 a6 64 a4  86
2421 : 65 20 ce 16 20 97 1b 20  78
2429 : f1 16 ad 00 0c 29 0f c9  6b
2431 : 02 90 e3 20 33 1b 4c ff  b5
2439 : 1a a9 03 a6 64 a4 65 20  ff
2441 : ce 16 20 f1 16 e6 67 4c  2f
2449 : 28 18 48 a9 01 20 d6 19  63
2451 : 68 29 0f a8 b9 68 1b 20  b2
2459 : 78 1b a9 03 20 d6 19 ad  a2
2461 : 01 0c 20 78 1b a9 04 20  cf
2469 : d6 19 ad 02 0c 20 78 1b  51
2471 : a9 0d 20 d2 ff ac 02 0c  89
2479 : b1 fd 29 fb 91 fd 60 00  7d
2481 : 00 14 15 16 17 18 19 1a  5e
2489 : 1b 1c 1d 00 00 00 4a 20  63
2491 : fb f9 a8 8a 20 d2 ff 98  ce
2499 : 4c d2 ff 8d 03 ff bd 00  27
24a1 : 80 8d 01 ff 9d 00 0b e8  00
24a9 : d0 f1 8e 00 ff 60 ad 03  d5
24b1 : d5 48 a5 ca 30 03 a9 3f  ad
24b9 : 2c a9 7f 8d 03 d5 a5 cb  58
24c1 : 8d e9 03 a2 00 20 e4 03  f3
24c9 : 68 8d 03 d5 60 a2 14 bd  5a
24d1 : 84 1b 9d e4 03 ca 10 f7  9d
24d9 : 60 a9 00 85 6c 85 70 a9  c7
24e1 : 05 20 d6 19 a9 09 20 d6  e0
24e9 : 19 20 db 18 a5 6b c9 4a  7e
24f1 : d0 04 a9 01 85 70 a9 06  dc
24f9 : 20 d6 19 20 db 18 a5 6b  bb
2501 : c9 4a d0 1f a9 07 20 d6  08
2509 : 19 58 20 cf ff c9 0d f0  b5
2511 : 0e 85 6e 20 cf ff 85 6f  73
2519 : a9 01 85 6c 4c 0c 1c a9  1b
2521 : 02 85 6c a9 08 20 d6 19  45
2529 : 58 20 cf ff c9 0d f0 26  9a
2531 : a2 01 86 6d ca 58 9d 00  89
2539 : 11 20 cf ff c9 0d f0 08  27
2541 : e8 e0 10 d0 f1 4c 43 1c  7e
2549 : a9 a0 e8 9d 00 11 e8 e0  1e
2551 : 10 90 f8 4c 43 1c a9 00  2d
2559 : 85 6d a9 0d 20 d2 ff 20  7a
2561 : d2 ff a9 00 20 d6 19 20  fb
2569 : db 18 78 60 d3 19 20 d8  b3
2571 : 18 78 60 19 20 d8 18 78  1b
2579 : 60 01 0c 85 12 ad 02 0c  bc
2581 : 85 13 a9 24 8d ac 02 a9  18
2589 : 01 85 0c 85 0d a9 00 8d  3a
2591 : b2 01 85 5e 85 ff 4c 00  7a
2599 : 0f a9 f0 85 3b 85 03 58  06
25a1 : a5 03 30 fc 78 6c fc ff  52
25a9 : a2 04 86 e7 a2 02 bd 00  1d
25b1 : 03 95 9b ca 10 f8 ad 01  40
25b9 : 04 85 06 ad 02 04 85 07  1c
25c1 : a9 4c 8d 00 03 a9 13 8d  d9
25c9 : 01 03 a9 05 8d 02 03 a6  99
25d1 : 06 e4 74 f0 03 a9 e0 2c  de
25d9 : a9 d0 85 00 58 a5 00 30  5f
25e1 : fc c9 02 b0 0b a6 06 86  64
25e9 : 74 09 80 8d 00 04 78 60  76
25f1 : a9 ff 85 74 c6 e7 f0 ed  d5
25f9 : a5 e7 c9 02 f0 22 a9 a1  4e
2601 : 8d 01 03 a9 05 8d 02 03  d0
2609 : a9 4c 8d 00 03 a9 e0 85  48
2611 : 00 a5 00 30 fc a5 16 00  3f
2619 : 00 00 00 00 00 00 00 a9  6d
2621 : 00 85 fa a9 13 85 fb a9  78
2629 : 18 85 fc a9 1c 85 fd a0  9f
2631 : 00 a2 0b b1 fc 91 fa c8  55
2639 : d0 f9 e6 fd e6 fb ca d0  9a
2641 : f2 4c 00 13 00 ff 00 ff  bc
```

Listing 1. Tornado-Copy 1571. Bitte verwenden Sie zur Eingabe den MSE auf Seite 109.

Der C 128 geht eigene Wege

Mit dem Aufbringen eines BOOT-Sektors auf eine Diskette kann man den C128 dazu veranlassen, bestimmte Programme nach dem Einschalten automatisch zu laden und zu starten. Das folgende Programm ermöglicht Ihnen die Herstellung eines solchen BOOT-Sektors.

Wenn man die Floppy-Station und anschließend den C128 einschaltet, dann fällt einem auf, daß das Diskettenlaufwerk anläuft und einen Zugriff durchführen will. Ist nun eine Diskette eingelegt, die einen speziellen BOOT-Sektor enthält, so wird automatisch ein bestimmtes Programm geladen und ausgeführt. Als Beispiel mag CP/M dienen.

Lag keine Diskette im Laufwerk, die einen solchen BOOT-Sektor enthält, so erfolgt der Start des Basic 7.0 des C128 und das Laufwerk hört mit dem Zugriff auf.

Ein solcher BOOT-Sektor hat einen bestimmten Aufbau (Bild 1). Wenn Sie nun gerne Ihre Disketten mit einem solchen Sektor versehen wollen, beispielsweise um Programme zu laden, ohne auch nur einen Finger zu rühren, dann kommt das Programm »Boseco« (Listing 1) gerade recht. Es erlaubt die Herstellung eines BOOT-Sektors und das Aufbringen auf eine Diskette. Das Wort BOOT-Sektor kommt übrigens vom Basic-Befehl BOOT, der anhand eines solchen Sektors Programme laden und starten kann.

Der BOOT-Sektor ist auf jeder Diskette der Sektor 0 auf Spur 1. Er enthält eine spezielle Meldung, die während des BOOTens ausgegeben wird und dazu noch den Namen des Programms, das geladen werden soll. Schließlich ist in dem Block auch noch ein Maschinenprogramm enthalten, das nach dem Laden alle weiteren Vorgänge steuern soll.

Das Programm Boseco wird geladen und mit RUN gestartet. Es kann nun der Text eingegeben werden, der während des BOOTens auf dem Bildschirm erscheinen soll, zum Beispiel: »Der Rest wird geladen...«

Anschließend gibt man die Anzahl der Programme an, die in den BOOT-Sektor geladen und (wenn er nicht ausreicht) auf dem Rest von Spur 1 der Diskette untergebracht werden sollen. Jetzt erwartet Boseco noch die Namen der einzelnen Programme, wobei die Filenamen angegeben werden müssen, unter denen die Programme auf einer Diskette gespeichert sind.

Nach der Angabe jedes Namens sind außerdem wichtige Daten über die einzelnen Maschinenprogramme an Boseco zu übergeben. Das ist zuerst einmal die Startadresse des entsprechenden Maschinenprogramms im Speicher des Computers. Diese ist dezimal anzugeben. Danach benötigt Boseco noch die Länge des jeweiligen Programms in Bytes – ebenfalls dezimal.

Sind diese Eingaben abgeschlossen, so sind den Aufforderungen entsprechend die Disketten mit den erforderlichen Maschinenprogrammen einzulegen, wobei mit <RETURN> quittiert wird. Boseco lädt dabei die einzelnen Programme in den Speicher, um sie anschließend auf dem BOOT-Sektor unterzubringen und diesen auf eine beliebige, formatierte Diskette zu schreiben.

Aufbau eines Boot-Sektors

Byte:	Bedeutung
\$00 - \$02	Identifizierung als Boot-Sektor (CBM)
\$03 - \$04	Adresse, ab der die Folgesektoren eingelesen werden.
\$05	Bank, in die die Folgesektoren eingelesen werden.
\$06	Anzahl der Folgesektoren.
\$07 bis Endekennung (\$00)	Text, der beim Booten auf dem Bildschirm erscheint.
bis Endekennung (\$00)	Dateinamen eines Programmes, das nachgeladen werden soll.
bis Byte \$FF	Maschinenprogramm

Bild 1. So ist ein BOOT-Sektor aufgebaut

Achtung: Es empfiehlt sich, BOOT-Sektoren nur auf leere Disketten zu schreiben und erst anschließend Programme auf diese Disketten zu speichern, da unter Umständen ein BOOT-Sektor-Bereich schon belegt sein kann. Läßt sich das nachträgliche Aufbringen eines BOOT-Sektors nicht vermeiden, so sollte man sich vor dem Start von Boseco vergewissern, daß die Spur 1 der Zieldiskette noch leer ist.

(O. Mangold/ks)

```
100 rem" boot sektor constructor
110 :
120 rem" by oliver mangold, laichingen
130 rem" dez. 1985
140 :
150 l=0:i=0:bl=0:t$="":x=0:y=0:le=0:l1=0:a$=""
:a=0:t=0:sl=0:hi=0:lo=0:pl=0:an=0
160 goto 500
170 :
180 rem" eingaberoutine
190 :
200 t$="":char,x,y:forl1=1tole:print".";:next
210 char,x,y,""
220 :
230 poke 2598,0:poke 2599,0
240 getkeya$
250 a=asc(a$)
260 if a=13 then print".";:poke 2599,1:return
270 if a=20 or a=157 then 370
280 if a>90 and a<193 then 230
290 if a<32 or a>218 then 230
300 t$=t$+a$:printa$;:le=le-1:x=x+1
310 if le>0 then 230
320 geta$:ifa$=chr$(13) then print" ";:poke 25
99,1:return
330 if a$=chr$(20) or a$=chr$(157) then 370
340 goto 320
350 :
360 :
370 poke 2599,1
380 if len(t$)=0 then 230
390 if le=0 then print" ";:goto 410
400 print ".";
410 le=le+1:x=x-1
420 t$=left$(t$,len(t$)-1)
430 char,x,y,"."
440 char,x,y
450 :
460 goto 230
470 :
```

Listing 1. Das Programm »Boseco«

```
480 rem" start hauprogramm
490 :
500 dim b%(110,256)
510 :
520 scnclr
530 char,15,2,chr$(18):print" BoSeCo ";chr$(14
6)
540 char,3,5,"Ein Programm zum Erstellen von"
550 char,3,7,"Boot - Sektoren."
560 char,3,10,"(w) by Oliver Mangold"
570 char,10,11,"Hagsbucherweg 20"
580 char,10,12,"7903 Laichingen"
590 char,10,13,"Tel.07333/4260
600 char,10,23,"Bitte Taste drücken."
610 getkeya$
620 :
630 rem" block 1/0 init
640 :
650 bl=1 : restore 1820
660 fori=1 to 6 : read b%(bl,i) :next
670 fori=1 to 13 : read b%(bl,67+i) : next
680 :
690 rem" 1. folgeblock init
700 :
710 bl=2 : restore 1870
720 fori=1 to 69 : read b%(bl,31+i) : next
730 :
740 scnclr
750 char,3,3,"Bitte geben Sie den Text, der"
760 char,3,5,"beim Booten angezeigt werden"
770 char,3,7,"soll, ein."
780 :
790 char,1,10,"Text :":y=12:x=1:le=38:gosub 20
0
800 :
810 rem" text in block 1/0 schreiben und rest
mit space auffüllen
820 :
830 bl=1:t$=chr$(13)+chr$(14)+chr$(150)+t$
840 l=len(t$)
850 fori=1to l :b%(bl,7+i)=asc(mid$(t$,i,1)) :
 next
860 if l=41 then 890
870 fori=l+8 to 48 : b%(bl,i)=32 : next
880 b%(bl,49)=13
890 :
900 b%(bl,51)=0:fori=52 to 67:b%(bl,i)=234:nex
t
910 :
920 scnclr
930 :
940 an=0
950 :
960 rem" prg-name und ladeadressen einlesen
970 :
980 scnclr
990 char,3,1,"Bitte geben Sie nun Programm-"
1000 char,3,3,"namen, Ladeadresse und Länge an
."
1010 char,0,6:y=5
1020 if an=5 then 1190
1030 print"   Name Prg.";:printan+1;":":x=17:y
=y+1:le=16:gosub200
1040 if t$="" then 1190
1050 an=an+1
1060 na$(an)=t$
1070 :
1080 print:print"   Ladeadresse :":x=17:y=y+1:
le=5:gosub200
1090 t=val(t$):nh(an)=int(t/256):nl(an)=t-256*
nh(an)
1100 print:print"   Länge        : ";:x=17:y=y+
1:le=5:gosub200
1110 la(an)=val(t$)
1120 :
1130 y=y+1
1140 print:print
1150 goto 1020
1160 :
1170 rem"programme einlesen und in folgeblöcke
 schreiben
1180 :
1190 scnclr:b%(2,1)=an:bl=2:pl=101:sl=0
1200 char,3,3,"Programme einlesen."
1210 k=6
1220 fori= 1 to an
1230 char,2,k,"Diskette für ":print na$(i)
1240 char,2,k+2,"einlegen und Return drücken."
1250 k=k+4
1260 :
1270 sl=sl+la(i-1):lo=8308+sl:hi=int(lo/256):l
o=lo-256*hi
1280 b%(2,6+i)=hi : b%(2,1+i)=lo
1290 getkeya$:if a$<>chr$(13) then 1290
1300 dopen#1,(na$(i))
1310 :
1320 get#1,a$:get#1,a$
1330 forj= 1 to la(i)
1340 get#1,a$:if a$=""thena$=chr$(0)
1350 b%(bl,pl)=asc(a$)
1360 pl=pl+1:if pl=257 then pl=1:bl=bl+1
1370 nextj
1380 :
1390 b%(2,16+i)=int(la(i)/256):b%(2,11+i)=la(i
)-256*b%(2,16+i)
1400 b%(2,21+i)=nl(i):b%(2,26+i)=nh(i)
1410 :
1420 dclose#1
1430 next i
1440 :
1450 rem" jsr aufrufe in block 1 integrieren
1460 :
1470 fori= 0 to an-1
1480 b%(1,i*3+81)=32  : rem" jsr
1490 b%(1,i*3+82)=b%(2,22+i) : rem" lo-byte
1500 b%(1,i*3+83)=b%(2,27+i) : rem" hi-byte
1510 next
1520 b%(1,i*3+81)=76:b%(1,i*3+82)=3:b%(1,i*3+8
3)=64
1530 :
1540 rem" anzahl der nachzuladenden blöcke in
block 1 schreiben
1550 :
1560 b%(1,7)=bl-1
1570 :
1580 rem" boot-sektoren auf disk schreiben
1590 :
1600 scnclr:char,5,10,"Bitte Diskette für Boot
-Sektor "
1610 char,5,12,"einlegen ."
1620 char,5,14,"Dann Taste drücken."
1630 getkeya$
1640 t=1:s=0
1650 open 15,8,15:open 2,8,2,"#"
1660 fori=1tobl
1670 print#15,"b-p 2 0"
1680 forj=1to256:print#2,chr$(b%(i,j));:next
1690 print#15,"u2 2 0";t;s
1700 print#15,"b-a 0";t;s
1710 s=s+1:if s=21 then t=t+1:s=0
1720 next
1730 :
1740 close2:close15
1750 char,5,20,ds$
1760 char,5,22,"Nochmal ...?"
1770 x=18:y=22:le=1:gosub200
1780 if t$<>"j" then end
1790 goto 1600
1800 rem" datas für kenncode,bootprogramm etc.
1810 :
1820 data 67,66,77
1830 data 16,32,0
1840 :
1850 data 169,80,141,0,10,169,11,141,1,10,32,4
7,32
1860 :
1870 data 206,16,32,48,63,173,16,32,168,185,17
,32,133,142,185,22,32,133,143
1880 data 185,32,32,170,185,37,32,133,158,185,
42,32,133,159,224,0,48,22,160,0
1890 data 177,142,145,158,136,208,249,230,143,
230,159,202,48,12,208
1900 data 240,173,16,32,168,185,27,32,168,208,
230,76,47,32,96
1910 :
```

Listing 1. Das Programm »Boseco« (Schluß)

»Zeichensetzereien« auf dem C 128

Mit »Zeichen-Fix« läßt sich der Zeichensatz des C 128 beliebig ändern. Somit sind internationale Zeichen kein Problem mehr. Die Tasten können wie bei professionellen Spielprogrammen mit kleinen Grafiken belegt werden, die dann später zusammen eine große Grafik ergeben.

Bei dem Programm »Zeichen-Fix« handelt es sich um einen Zeichensatz-Generator für den C 128. Mit diesem Programm ist es möglich, neben dem Zeichensatz des Computers auch eigene Zeichen zu kreieren, um damit entweder Sonderzeichen oder Grafiken behandeln zu können.

Nach dem Start von Zeichen-Fix muß man 15 Sekunden warten, bis der Original-Zeichensatz ins RAM (Startadresse: 8192) kopiert worden ist. Im Hauptmenü sind anschließend folgende Optionen verfügbar:

1. Zeichen verändern

Dies ist der wichtigste Teil des Programms. Hier können die neuen Zeichen erstellt werden.

Man wird aufgefordert das Zeichen einzutippen, das verändert werden soll (auch SHIFT-Kombinationen sind möglich). Daraufhin erscheint das Zeichen in einer 8x8-Matrix. Mit der <CLR/HOME>-Taste wird der Cursor in die linke obere Ecke gesetzt. Mit <SHIFT+CLR/HOME> wird die Matrix gelöscht. Beim Drücken von <I> wird die Matrix invertiert. Die <D>-Taste dreht das Zeichen um 180 Grad. Bewegt wird der Cursor mit den Cursor-Tasten. Punkte werden nach Druck auf die <*>-Taste gesetzt, gelöscht werden sie mit der Leertaste. Um ins Hauptmenü zu kommen, muß die Taste <←> gedrückt werden. Ist ein Zeichen fertiggestellt, muß es mit der <RETURN>-Taste in den neuen Zeichensatz aufgenommen werden.

Eine Übersicht dieser Befehle finden Sie auch auf dem Bildschirm links neben der Matrix.

2. Zeichensatz speichern

Unter diesem Menüpunkt können fertige Zeichensätze auf Diskette gespeichert werden. Ein Zeichensatz belegt neun Blöcke auf der Diskette.

3. Zeichensatz laden

Das Laden von Zeichensätzen ist mit diesem Menüpunkt möglich. Der Zeichensatz kann dann weiter bearbeitet oder mit Menüpunkt 4 aktiviert werden.

4. Zeichensatz aktivieren

Der erstellte Zeichensatz wird aktiviert, das heißt gestartet, und das Programm verlassen. Vorher wird die <RESTORE>-Taste außer Funktion gesetzt, weil nach Drücken von <RUN/STOP+RESTORE> ansonsten der Original-Zeichensatz aktiviert würde.

Die <RESTORE>-Taste kann mit POKE 792,64 wieder eingeschaltet werden.

5. Directory

Bei dieser Funktion wird das Directory der eingelegten Diskette auf den Bildschirm gebracht. Mit einem Tastendruck kommt man wieder ins Hauptmenü.

6. Ende

Das Programm wird beendet.

Erstellte Zeichensätze können auch ohne Zeichen-Fix verwendet werden:

Zuerst muß man die Grafikseite mit GRAPHIC 1,1:GRAPHIC 0 löschen. Danach ist der Zeichensatz mit BLOAD-"NAME",ON B0,P8192 zu laden. Aktiviert wird er mit POKE 2604, (PEEK(2604) AND 240) OR 8. Ein Deaktivieren ist mit POKE 2604, (PEEK(2604) AND 240) OR 4 möglich.

Beim Abtippen des Programms (Listing 1) ist zu beachten, daß in der Zeile 550 die Grafikzeichen einzusetzen sind, die mit <CBM+Y> erreicht und in der Zeile 460 die, die mit <CBM+P> erreicht werden.

In den Zeilen 470 bis 540 müssen jeweils die zweiten Grafikzeichen mit <CBM+N> und die dritten Grafikzeichen mit <CBM+H> eingetippt werden. Außerdem sollte man darauf achten, daß in Programmen, in denen der neue Zeichensatz benutzt wird, nicht zusätzlich hochauflösende Grafik verwendet wird, da beim Löschen des Grafikspeichers auch der Zeichensatz gelöscht wird.

Programmbeschreibung:

In den Zeilen 80 bis 160 wird der Original-Zeichensatz ins RAM kopiert.

Die Zeilen 170 bis 690 erstellen das Hauptmenü und die Matrix zum Erstellen von Zeichen. Außerdem wird in der Zeile 370 festgelegt, wohin je nach Wahl aus dem Hauptmenü gesprungen werden soll.

Die Zeilen 700 bis 790 holen das vorher gewählte Zeichen in die Matrix.

Ab der Zeile 800 beginnt dann die Routine zum Verändern der Zeichen.

In den Zeilen 810 bis 940 erfolgt die Abfrage, in welches Unterprogramm, je nach Tastendruck, gesprungen werden soll.

Die Zeilen 890 bis 1230 übernehmen die Steuerung des Cursors.

Ab 1240 folgen dann Sonderfunktionen, wie Punkte setzen und löschen, Cursor in die obere linke Ecke (HOME), Matrix löschen (CLR), Matrix invertieren, Zeichen in Zeichensatz übernehmen (RETURN), ins Hauptmenü wechseln (←) und Zeichen drehen.

Danach folgen in den Zeilen 2100 bis 2200 die Diskettenfunktionen Laden und Speichern und von 2210 bis 2240 die Aktivierungsroutine für die Zeichensätze.

Zum Schluß folgt ab 2250 eine Fehlerabfang-Routine. Diese Routine fängt alle Fehler, auch Diskettenfehler, ab.

(D. Koller/ks)

```
10 REM ************************************
20 REM *****      ZEICHEN-FIX        *****
30 REM *****      FUER DEN PC 128    *****
40 REM ***** VON DIRK KOLLER         *****
50 REM *****      6305 BUSECK 1      *****
60 REM ************************************
70 MA=8192: MO=53248: ADR=1533
80 REM *** ZEICHEN I. RAM KOPIEREN ***
90 GRAPHIC 1,1
100 GRAPHIC 0
110 FAST
120 BANK 14
130 FOR I=.TO 2048
140 POKE(MA+I),PEEK(MO+I)
150 NEXT
160 SLOW
170 REM *** HAUPTMENU ****************
180 TRAP 2250
190 COLOR 0,16: COLOR 4,16: COLOR 5,1
200 SCNCLR
```

Listing 1. Der »Zeichen-Fix«-Editor für den C 128




PROTEXT



```
210 CHAR 1,10,2,"@@@@@@@@@@@@@@@@@@@@@@"
220 CHAR 1,10,3,"G{20SPACE}M"
230 CHAR 1,10,4,"G ZEICHEN-FIX PC 128 M"
240 CHAR 1,10,5,"G{2SPACE}VON DIRK KOLLER{3SPACE
    }M"
250 CHAR 1,10,6,"G{20SPACE}M"
260 CHAR 1,10,7,"TTTTTTTTTTTTTTTTTTTTTT"
270 CHAR 1,7,10,"1 : NEUE ZEICHEN ERSTELLEN"
280 CHAR 1,7,12,"2 : ZEICHENSATZ ABSPEICHERN"
290 CHAR 1,7,14,"3 : ZEICHENSATZ LADEN"
300 CHAR 1,7,16,"4 : ZEICHENSATZ AKTIVIEREN"
310 CHAR 1,7,18,"5 : DIRECTORY"
320 CHAR 1,7,20,"6 : ENDE"
330 GET KEY NU$
340 NU=VAL(NU$)
350 IF NU<1 OR NU>6 THEN PRINT CHR$(7): GOTO 170
360 IF NU=6 THEN END
370 ON NU GOTO 380,2100,2160,2210,2320
380 REM *** NEUE ZEICHEN ERSTELLEN ****
390 SCNCLR
400 CHAR 1,10,3,"ZEICHEN ERSTELLEN"
410 CHAR 1,10,4,"TTTTTTTTTTTTTTTTT"
420 CHAR 1,0,7,"BEFEHLE:"
430 CHAR 1,0,8,"TTTTTTT"
440 CHAR 1,0,9,"-HOME :CURSOR LINKS OBEN{3SPACE}
    K{10SPACE}"
450 CHAR 1,0,10,"-CLR{2SPACE}:LOESCHT MATRIX{6SP
    ACE}K MATRIX: "
460 CHAR 1,0,11,"-I{4SPACE}:INVERTIERT MATRIX{3S
    PACE}K PPPPPPPP"
470 CHAR 1,0,12,"-D{4SPACE}:DREHT ZEICHEN{7SPACE
    }KN{8SPACE}M"
480 CHAR 1,0,13,"-*{4SPACE}:PUNKT SETZEN{8SPACE}
    KN{8SPACE}G"
490 CHAR 1,0,14,"-SPACE:PUNKT LOESCHEN{6SPACE}KN
    {8SPACE}M"
500 CHAR 1,0,15,"-PFEILTASTEN BEWEGEN DEN{3SPACE
    }KN{8SPACE}M"
510 CHAR 1,0,16," CURSOR UEBER DIE MATRIX{3SPACE
    }KN{8SPACE}M"
520 CHAR 1,0,17,"-RETURN:SICHERT ZEICHEN{4SPACE}
    KN{8SPACE}M"
530 CHAR 1,0,18,"-←{4SPACE}:ZURUECK INS MENU{4SP
    ACE}KN{8SPACE}M"
540 CHAR 1,27,19,"KN{8SPACE}M"
550 CHAR 1,27,20,"{2SPACE}YYYYYYYY "
560 CHAR 1,0,21,"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
    YYYYYYYY"
570 CHAR 1,5,23,"WELCHES ZEICHEN SOLL VERAENDERT
    ",1
580 CHAR 1,5,24,"WERDEN ? ",1
590 GET KEY A$
600 TE=ASC(A$)
610 IF TE<33 OR TE>255 THEN PRINT "{UP}" CHR$(7)
    : GOTO 570
620 IF TE<160 AND TE>127 THEN PRINT "{UP}" CHR$(
    7): GOTO 570
630 PRINT A$;
640 CHAR 1,5,23,"{38SPACE}"
650 CHAR 1,5,24,"{12SPACE}"
660 IF TE AND 128 THEN CO=TE AND 127 OR 64: GOTO
     700
670 IF NOT TE AND 64 THEN CO=TE: GOTO 700
680 IF TE AND 32 THEN CO=TE AND 95: GOTO 700
690 CO=TE AND 63
700 REM *** ZEICHEN IN MATRIX ********
710 AS=MA+CO*8
720 FOR Z=0 TO 7
730 FOR S=0 TO 7
740 BY=PEEK(AS+Z) AND 2↑(7-S)
750 ZE$=" "
760 IF BY<>0 THEN ZE$="*"
770 CHAR 1,29+S,12+Z,ZE$
780 NEXT S
790 NEXT Z
800 REM *** ZEICHEN VERAENDERN *******
810 AZEI=PEEK(ADR)
820 POKE ADR,81
830 GET KEY OP$
840 IF OP$="{RIGHT}" OR OP$="{LEFT}" OR OP$="{UP
    }" OR OP$="{DOWN}" THEN 890
850 IF OP$="{HOME}" OR OP$="{CLR}" OR OP$="I" OR
     OP$="D" THEN 1280
860 IF OP$=CHR$(13) OR OP$="←" THEN 1670
870 IF OP$="*" OR OP$=" " THEN 1240
880 PRINT CHR$(7)"{UP}": GOTO 830
890 REM *** CURSOR BEWEGEN ***********
900 IF OP$="{RIGHT}" THEN GOSUB 950
910 IF OP$="{LEFT}" THEN GOSUB 1030
920 IF OP$="{UP}" THEN GOSUB 1100
930 IF OP$="{DOWN}" THEN GOSUB 1170
940 GOTO 830
950 REM *** CURSOR RECHTS ************
960 IF PEEK(ADR+1)=116 OR PEEK(ADR+1)=101 THEN R
    ETURN
970 POKE ADR,AZEI
980 ADR=ADR+1
990 AZEI=PEEK(ADR)
1000 POKE ADR,81
1010 RETURN
1020 POKE ADR,81
1030 REM *** CURSOR LINKS ************
1040 IF PEEK(ADR-1)=103 OR PEEK(ADR-1)=106 THEN
     RETURN
1050 POKE ADR,AZEI
1060 ADR=ADR-1
1070 AZEI=PEEK(ADR)
1080 POKE ADR,81
1090 RETURN
1100 REM *** CURSOR HOCH *************
1110 IF PEEK(ADR-40)=111 THEN RETURN
1120 POKE ADR,AZEI
1130 ADR=ADR-40
1140 AZEI=PEEK(ADR)
1150 POKE ADR,81
1160 RETURN
1170 REM *** CURSOR RUNTER ***********
1180 IF PEEK(ADR+40)=119 THEN RETURN
1190 POKE ADR,AZEI
1200 ADR=ADR+40
1210 AZEI=PEEK(ADR)
1220 POKE ADR,81
1230 RETURN
1240 REM *** PUNKT SETZEN/LOESCHEN ***
1250 IF OP$="*" THEN AZEI=42
1260 IF OP$=" " THEN AZEI=32
1270 GOTO 830
1280 REM *** CLR,HOME,I & D **********
1290 IF OP$="{HOME}" THEN GOSUB 1340
1300 IF OP$="{CLR}" THEN GOSUB 1400
1310 IF OP$="I" THEN GOSUB 1520
1320 IF OP$="D" THEN GOSUB 1900
1330 GOTO 830
1340 REM *** HOME ********************
1350 POKE ADR,AZEI
1360 ADR=1533
1370 AZEI=PEEK(ADR)
1380 POKE ADR,81
1390 RETURN
1400 REM *** CLEAR *******************
1410 CHAR 1,10,23,"EINEN MOMENT BITTE ..."
1420 FOR ZEI =0 TO 7
1430 FOR SPA =0 TO 7
1440 CHAR 1,29+SPA,12+ZEI," "
1450 NEXT SPA
1460 NEXT ZEI
1470 ADR=1533
1480 AZEI=PEEK(ADR)
1490 POKE ADR,81
1500 CHAR 1,10,23,"{38SPACE}"
1510 RETURN
1520 REM *** MATRIX INVERTIEREN ******
1530 CHAR 1,10,23,"EINEN MOMENT BITTE ..."
1540 POKE ADR,AZEI
1550 FOR I=0 TO 7
1560 FOR B=0 TO 7
1570 INV=1533+(B+(I*40))
1580 IF PEEK(INV)=42 THEN POKE INV,32: GOTO 1600
1590 IF PEEK(INV)=32 THEN POKE INV,42
1600 NEXT B
1610 NEXT I
1620 ADR=1533
1630 AZEI=PEEK(ADR)
1640 POKE ADR,81
1650 CHAR 1,10,23,"{38SPACE}"
1660 RETURN
1670 REM *** RETURN & ← **************
1680 IF OP$=CHR$(13) THEN GOSUB 1710
1690 IF OP$="←" THEN 170
1700 GOTO 830
1710 REM *** ZEICHEN IN RAM POKEN ****
1720 FOR LO=0 TO 7: WE(LO)=0: NEXT LO
```

Listing 1. Der »Zeichen-Fix«-Editor für den C 128 (Forts.)

```
1730 CHAR 1,10,23,"EINEN MOMENT BITTE ..."
1740 POKE ADR,AZEI
1750 FOR I=0 TO 7
1760 FOR B=0 TO 7
1770 INV=1533+(B+(I*40))
1780 IF PEEK(INV)=42 THEN WE(I)=WE(I)+(128/(2↑B))
1790 NEXT B
1800 NEXT I
1810 PO=MA+CO*8
1820 FOR I=0 TO 7
1830 POKE PO+I,WE(I)
1840 NEXT I
1850 ADR=1533
1860 AZEI=PEEK(ADR)
1870 POKE ADR,81
1880 CHAR 1,10,23,"{38SPACE}"
1890 RETURN
1900 REM *** ZEICHEN DREHEN **********
1910 CHAR 1,10,23,"EINEN MOMENT BITTE ..."
1920 FOR I=0 TO 7: WE(I)=0: NEXT I
1930 POKE ADR,AZEI
1940 FOR I=0 TO 7
1950 FOR B=0 TO 7
1960 INV=1533+(B+(I*40))
1970 IF PEEK(INV)=42 THEN WE(I)=WE(I)+(128/(2↑B)
)
1980 NEXT B
1990 NEXT I
2000 AUSH=WE(0): WE(0)=WE(7): WE(7)=AUSH
2010 AUSH=WE(1): WE(1)=WE(6): WE(6)=AUSH
2020 AUSH=WE(2): WE(2)=WE(5): WE(5)=AUSH
2030 AUSH=WE(3): WE(3)=WE(4): WE(4)=AUSH
2040 PO=MA+CO*8
2050 FOR I=0 TO 7
2060 POKE PO+I,WE(I)
2070 NEXT I
2080 CHAR 1,10,23,"{38SPACE}"
2090 GOTO 700
2100 REM *** ZEICHENSATZ SPEICHERN ***
2110 CHAR 1,0,22,"NAME DES ZEICHENSATZES{6SPACE}
"
2120 INPUT " {5LEFT}";NA$
2130 BSAVE(NA$),ON B0,P8192 TO P10240
2140 CHAR 1,0,22,"{38SPACE}"
2150 GOTO 170
2160 REM *** ZEICHENSATZ LADEN *******
2170 CHAR 1,0,22,"NAME DES ZEICHENSATZES "
2180 INPUT "{6SPACE,7LEFT}";NA$
2190 BLOAD(NA$),ON B0,P8192
2200 GOTO 170
2210 REM *** ZEICHENSATZ AKTIVIEREN **
2220 POKE 792,98
2230 POKE 2604,(PEEK(2604) AND 240) OR 8
2240 END
2250 V=ER
2260 V$=ERR$(V)
2270 CHAR 1,0,22,"{38SPACE}"
2280 CHAR 1,5,22,V$
2290 CHAR 1,0,23,"{5SPACE}FEHLER BEHEBEN - TASTE
 DRUECKEN"
2300 GET KEY WA$
2310 RESUME 170
2320 REM *** DIRECTORY ***************
2330 SCNCLR
2340 CATALOG
2350 GET KEY DU$
2360 GOTO 170
```

Listing 1. Der »Zeichen-Fix«-Editor für den C128 (Schluß)

Centronics-Schnittstelle für den C 128

Wir machen kurzen Prozeß mit den Anschlußproblemen Ihres Druckers! Mit einem leicht herstellbaren Kabel und unserem Programm können Sie mit nahezu allen Druckern problemlos am C128 arbeiten.

Der C128 besitzt mit dem Basic 7.0 einen recht umfangreichen Befehlswortschatz, der auch die hochauflösende Grafikprogrammierung unterstützt. Doch was nützt die schönste Grafik auf dem Bildschirm, wenn man sie nicht auch auf einem Drucker ausgeben kann? Wenn Sie einen Drucker mit Centronics-Schnittstelle besitzen, gehören Sie zu den Menschen, die dieses Problem nicht länger betrifft. Der abgedruckte Druckertreiber enthält eine Routine zur Umwandlung von CBM-ASCII nach Standard-ASCII, so daß im Textmodus nun weiterhin auch die deutschen Umlaute wie jedes andere Zeichen verwendet und ausgedruckt werden können. CBM-Sonder- und Grafikzeichen können jedoch nicht auf dem Drucker realisiert werden. Es ist deshalb ratsam, derartige Zeichen in eigenen Listings als CHR$(Zeichen) darzustellen.

Der Centronics-Druckertreiber für den C128 (Listing 1) liegt im freien RAM-Bereich von $1300 bis $17E3 und wird mit SYS 4864 aktiviert. Hierbei wird die Betriebssystemroutine zur Ausgabe von Zeichen auf den Treiber umgelenkt. Nach <RUN/STOP + RESTORE> oder einem Reset muß das Programm erneut aktiviert werden.

Das Treiberprogramm wird über die Sekundäradresse gesteuert. Bei Sekundäradresse Null wird der Text ohne ASCII-Wandlung ausgegeben; bei Sekundäradresse 1 bis 14 erfolgt eine Zeichensatzanpassung über eine 256 Byte lange Tabelle ($142a bis $1529), die auch nach eigenen Bedürfnissen angepaßt werden kann. Die Sekundäradresse 15 ist reserviert für die Ausgabe einer hochauflösenden Grafik. Durch Übertragung der Ziffern 1 oder 2 kann die Größe der Hardcopy bestimmt werden (siehe Beispiel-Listing 2 und Bild 1). Die größtmögliche Hardcopy (Ziffer 2) entspricht ungefähr DIN A5. Durch mehrstellige Ziffern kann die Hardcopy beliebig oft wiederholt werden. Vor jeder Hardcopy wird zur Druckjustage eine Leerzeile gedruckt.

Anpassungen

Abhängig vom Drucker oder der individuellen Anforderung des Benutzers, können einige Voreinstellungen des Treibers durch Modifikation, zum Beispiel mit dem eingebauten Maschinensprachemonitor, geändert werden. Die entsprechenden Speicherstellen und die Bedeutung der voreingestellten Inhalte sind nachfolgend beschrieben.

Speicherstelle $1303 (dezimal 4867):

In dieser Speicherstelle ist die Gerätenummer abgelegt. Der Ursprungswert ist #4. Soll ein Drucker mit der Gerätenummer #5 angesprochen werden, so muß der Inhalt dieser Adresse entsprechend geändert werden.

Test der kleinen Hardcopy

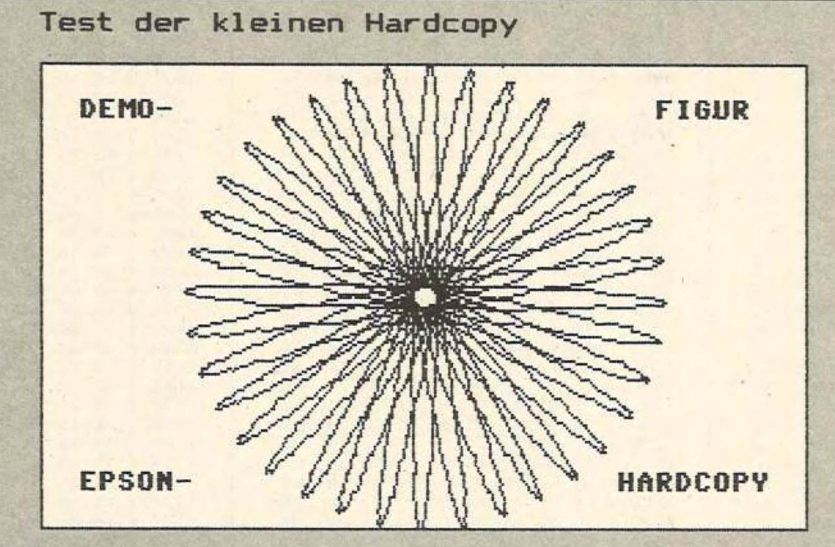


Bild 1. Mit Listing 2 können diese beiden Hardcopies erstellt und ausgedruckt werden

Test der grossen Hardcopy

DEMO- FIGUR

EPSON- HARDCOPY

Speicherstelle $1304 (dezimal 4868)

Durch den Inhalt dieser Speicherstelle wird festgelegt, ob am Zeilenende Carriage Return (CR = Wagenrücklauf) gesendet wird (Inhalt = 0) oder Carriage Return mit Line Feed (LF = Zeilenvorschub) (Inhalt = 1). Voreingestellt ist CR und LF.

Speicherstellen $142a bis $1529 (dezimal 5162 bis 5417)

In diesem Speicherbereich befindet sich die Umwandlungstabelle für die Anpassung von CBM-ASCII nach Standard-ASCII. Die hier aufgeführten Werte werden bei Sekundäradresse ungleich Null anstelle der üblichen Codes an den Drucker gesendet. Zur Darstellung der Umlaute auf einem Epson-Drucker müssen die Codes über diese Tabelle gewandelt werden. Die richtigen Codes sind hierin bereits enthalten.

Speicherstellen $1540 bis $1544 (dezimal 5440 bis 5444)

Hier befindet sich die für Initialisierung der kleineren Standard-Hardcopy wichtige Escape-Sequenz:

1b 2a 04 40 01 (27, '*', 4, <320,>320)

Speicherstellen $1546 bis $154A (dezimal 5446 bis 5450)

In diesem Bereich ist die entsprechende Escape-Sequenz für die vergrößerte Hardcopy-Routine enthalten:

1b 2a 06 80 02 (27, '*', 6, <640, >640)

Speicherstelle $156f (dezimal 5487)

In diesem Register ist der Randabstand für die Standard-Hardcopy enthalten (Voreinstellung #20).

```
Name : druckertr. 128     1300 17e4
-----------------------------------
1300 : 4c 05 13 04 01 a2 71 8e   1c
1308 : 1a 03 a2 13 8e 1b 03 a2   c2
1310 : a3 8e 1c 03 a2 13 8e 1d   99
1318 : 03 a2 bc 8e 1e 03 a2 13   18
1320 : 8e 1f 03 a2 d4 8e 20 03   9b
1328 : a2 13 8e 21 03 a2 ec 8e   32
1330 : 26 03 a2 13 8e 27 03 78   02
1338 : a9 ff 8d 03 dd ad 02 dd   b4
1340 : 09 04 8d 02 dd ad 00 dd   f6
1348 : 09 04 8d 00 dd a9 10 8d   3d
1350 : 0d dd ad 0d dd 58 60 48   0c
1358 : 8d 01 dd ad 00 dd 29 fb   1e
1360 : 8d 00 dd 09 04 8d 00 dd   ee
1368 : ad 0d dd 29 10 f0 f9 68   7a
1370 : 60 a6 b8 20 02 f2 d0 03   56
1378 : 4c 7f f6 a6 98 e0 0a 90   f0
1380 : 03 4c 7c f6 e6 98 a5 b8   e2
1388 : 9d 62 03 a5 b9 09 60 9d   6c
1390 : 76 03 a5 ba 9d 6c 03 cd   2d
1398 : 03 13 d0 02 18 60 c9 00   45
13a0 : 4c df ef 20 07 f2 f0 02   ac
13a8 : 18 60 20 12 f2 8a 48 a5   2b
13b0 : ba cd 03 13 f0 03 4c 94   f6
13b8 : f1 4c e4 f1 20 02 f2 f0   06
13c0 : 03 4c 82 f6 20 12 f2 a5   12
13c8 : ba cd 03 13 d0 03 4c 8b   f9
13d0 : f6 4c 10 f1 20 02 f2 f0   ee
13d8 : 03 4c 82 f6 20 12 f2 a5   2a
13e0 : ba cd 03 13 d0 03 4c 69   cd
13e8 : f1 4c 56 f1 8d 2a 15 48   e2
13f0 : a5 9a cd 03 13 f0 03 4c   13
13f8 : 7c ef 86 97 a5 b9 29 0f   eb
1400 : c9 0f d0 03 4c 2b 15 c9   eb
1408 : 00 f0 06 68 aa bd 2a 14   78
1410 : 48 68 20 57 13 ae 04 13   5c
1418 : f0 09 c9 0d d0 05 a9 0a   91
1420 : 20 57 13 a6 97 ad 2a 15   3f
1428 : 18 60 00 01 02 03 04 05   e3
1430 : 06 07 08 09 0a 0b 0c 0d   20
1438 : 0e 0f 10 11 12 13 14 15   28
1440 : 16 17 18 19 1a 1b 1c 1d   30
1448 : 1e 1f 20 21 22 23 24 25   38
1450 : 26 27 28 29 2a 2b 2c 2d   40
1458 : 2e 2f 30 31 32 33 34 35   48
1460 : 36 37 38 39 3a 3b 3c 3d   50
1468 : 3e 3f 40 61 62 63 64 65   20
1470 : 66 67 68 69 6a 6b 6c 6d   60
1478 : 6e 6f 70 71 72 73 74 75   68
1480 : 76 77 78 79 7a 5b 5c 5d   ae
1488 : 5e 5f 60 61 62 63 64 65   78
1490 : 66 67 68 69 6a 6b 6c 6d   80
1498 : 6e 6f 70 71 72 73 74 75   88
14a0 : 76 77 78 79 7a 7b 7c 7d   90
14a8 : 7e 7f 80 81 82 83 84 85   97
14b0 : 86 87 88 89 8a 8b 8c 8d   9f
14b8 : 8e 8f 90 91 92 93 94 95   a7
14c0 : 96 97 98 99 9a 9b 9c 9d   af
14c8 : 9e 9f a0 a1 a2 a3 a4 a5   b7
14d0 : a6 a7 a8 a9 aa ab ac ad   bf
14d8 : ae af b0 b1 b2 b3 b4 b5   c7
14e0 : b6 b7 b8 b9 ba 7b 7c 7d   4c
14e8 : 7e bf c0 41 42 43 44 45   78
14f0 : 46 47 48 49 4a 4b 4c 4d   e0
14f8 : 4e 4f 50 51 52 53 54 55   e8
1500 : 56 57 58 59 5a 5b 5c 5d   f0
1508 : de df e0 e1 e2 e3 e4 e5   f7
1510 : e6 e7 e8 e9 ea eb ec ed   ff
1518 : ee ef f0 f1 f2 f3 f4 f5   07
1520 : f6 f7 f8 f9 fa fb fc fd   0f
1528 : fe ff 00 68 38 e9 30 8d   e2
1530 : 68 15 20 7a 15 18 a6 97   56
1538 : ad 2a 15 60 03 1b 40 06   62
1540 : 1b 2a 04 40 01 06 1b 2a   7b
1548 : 06 80 02 03 1b 31 0d 0a   f3
1550 : 06 ad 6c 08 20 90 28 ad   cb
1558 : 71 08 c9 17 f0 36 c9 19   3d
1560 : f0 45 a9 09 2d 75 08 d0   bf
1568 : 01 00 20 00 00 80 19 14   02
1570 : 07 00 00 00 00 00 00 00   78
1578 : 28 02 a2 01 bd 4b 15 20   35
1580 : 57 13 e8 ec 4b 15 d0 f4   c3
1588 : ae 68 15 e0 01 d0 03 4c   07
1590 : 2b 17 e0 02 d0 03 4c 9d   51
1598 : 15 20 83 17 60 20 b4 17   89
15a0 : ad 69 15 8d 6b 15 ad 6a   e4
15a8 : 15 8d 6c 15 a9 19 8d 6e   b8
15b0 : 15 ae 6e 15 20 be 15 ca   3c
15b8 : d0 fa 20 83 17 60 20 d9   27
15c0 : 16 ad 6b 15 85 8b ad 6c   6f
15c8 : 15 85 8c 20 f5 15 18 ad   8b
15d0 : 6b 15 69 04 85 8b ad 6c   e5
15d8 : 15 69 00 85 8c 20 f5 15   1e
15e0 : 18 a9 40 6d 6b 15 8d 6b   f7
15e8 : 15 a9 01 6d 6c 15 8d 6c   3e
15f0 : 15 20 06 17 60 20 d9 16   14
15f8 : 20 98 17 a2 01 bd 45 15   bc
1600 : 20 57 13 e8 ec 45 15 d0   9d
1608 : f4 ac 78 15 20 29 16 20   f7
1610 : 81 16 18 a5 8b 69 08 85   86
1618 : 8b a5 8c 69 00 85 8c 88   36
1620 : d0 ea 20 b4 17 20 06 17   bd
1628 : 60 20 d9 16 a2 00 a0 00   7e
1630 : a9 08 8d 71 15 b1 8b 8d   97
1638 : 74 15 ad 74 15 2a 8d 74   f2
1640 : 15 a9 00 6a 8d 73 15 a9   93
1648 : 02 8d 72 15 ad 73 15 2a   6f
1650 : bd 60 15 2a 9d 60 15 bd   75
1658 : 58 15 2a 9d 58 15 ce 72   c7
1660 : 15 d0 e9 ce 71 15 d0 d2   da
1668 : bd 60 15 e8 9d 60 15 ca   7e
1670 : bd 58 15 e8 9d 58 15 e8   7e
1678 : c8 c0 04 d0 b3 20 06 17   3e
1680 : 60 20 d9 16 a9 58 85 8d   b8
1688 : a9 15 85 8e 20 a4 16 20   af
1690 : d0 17 a9 60 85 8d a9 15   f8
1698 : 85 8e 20 a4 16 20 d0 17   d5
16a0 : 20 06 17 60 20 d9 16 a2   04
16a8 : 00 a9 80 8d 6d 15 a0 00   51
16b0 : b1 8d 2c 6d 15 f0 05 d0   6f
16b8 : 00 38 b0 01 18 bd 50 15   fb
16c0 : 2a 9d 50 15 c8 c0 08 d0   c4
16c8 : e7 ad 6d 15 4a 8d 6d 15   75
16d0 : e8 e0 08 d0 d9 20 06 17   29
16d8 : 60 08 8d 74 15 68 8d 75   e4
16e0 : 15 68 8d 76 15 68 8d 77   15
16e8 : 15 ad 75 15 48 ad 74 15   c2
16f0 : 48 8a 48 98 48 ad 77 15   9c
16f8 : 48 ad 76 15 48 ad 75 15   49
1700 : 48 ad 74 15 28 60 68 8d   21
1708 : 76 15 68 8d 77 15 68 a8   e8
1710 : 68 aa 68 8d 74 15 68 8d   46
1718 : 75 15 ad 77 15 48 ad 76   a9
1720 : 15 48 ad 75 15 48 ad 74   a7
1728 : 15 28 60 20 b4 17 ad 69   fb
1730 : 15 85 8d ad 6a 15 85 8e   a3
1738 : a9 19 8d 6e 15 a9 28 8d   f9
1740 : 78 15 ae 6e 15 20 98 17   9f
1748 : 20 52 17 ca d0 f7 20 83   05
1750 : 17 60 20 d9 16 a2 01 bd   d1
1758 : 3f 15 20 57 13 e8 ec 3f   bf
1760 : 15 d0 f4 ae 78 15 20 a4   ea
1768 : 16 18 a5 8d 69 08 85 8d   ad
1770 : a5 8e 69 00 85 8e 20 d0   a6
1778 : 17 ca d0 ea 20 b4 17 20   ca
1780 : 06 17 60 20 d9 16 a2 01   09
1788 : bd 3c 15 20 57 13 e8 ec   38
1790 : 3c 15 d0 f4 20 06 17 60   79
1798 : 20 d9 16 ae 68 15 ca bd   d6
17a0 : 6f 15 aa a9 20 e0 00 f0   65
17a8 : 07 20 57 13 ca 4c a5 17   cb
17b0 : 20 06 17 60 20 d9 16 ae   2c
17b8 : 04 13 e8 8e 79 15 a2 00   1d
17c0 : bd 4e 15 20 57 13 e8 ec   79
17c8 : 79 15 d0 f4 20 06 17 60   ee
17d0 : 20 d9 16 a2 00 bd 50 15   10
17d8 : 20 57 13 e8 e0 08 d0 f5   03
17e0 : 20 06 17 60 bf 40 bf 40   53
```

Listing 1. Centronics-Druckertreiber für den C128. Bitte geben Sie das Programm im C64-Modus mit dem MSE auf Seite 109 ein.

```
10 REM GRAFIK UND HARDCOPYTEST
20 COLOR 0,2: COLOR 1,7: COLOR 4,15
30 GRAPHIC 1,1
40 BOX 1,0,0,319,199
50 FOR I=1 TO 180 STEP 10
60 CIRCLE 1,160,100,100,5,,,I
70 NEXT I
80 CHAR 1,2,2,"DEMO-"
90 CHAR 1,32,2,"FIGUR"
100 CHAR 1,2,22,"EPSON-"
110 CHAR 1,30,22,"HARDCOPY
120 OPEN 1,4,1
130 PRINT#1,TAB(20);"TEST DER KLEINEN HARDCO
   PY"
140 OPEN 2,4,15
150 PRINT#2,1
160 PRINT#1,
170 PRINT#1,TAB(7);"TEST DER GROSSEN HARDCOP
   Y"
180 PRINT#2,2
190 CLOSE 1: CLOSE 2
200 GRAPHIC 0
```

Listing 2. Programm zur Demonstration der beiden integrierten Hardcopy-Routinen.

A	GND	16
B	Flag-Busy	11 oder
	Flag-Acknowl.	10
C	D0	2
D	D1	3
E	D2	4
F	D3	5
H	D4	6
J	D5	7
K	D6	8
L	D7	9
M	PA2-Strobe	1

Bild 2. Anschlußbelegung für ein Parallelkabel zwischen dem Userport des C128 und einem Centronics-Drucker

Speicherstelle $1570 (dezimal 5488)

Der Randabstand für die vergrößerte Hardcopy ist in dieser Speicherstelle mit #7 voreingestellt.

Wollen Sie einen Drucker anschließen, der keine Grafik-Hardcopy beherrscht, können Sie die Hardcopy-Routinen entfernen. Dazu ist in Adresse $1401 der Wert $0f (dezimal 15) in $10 (dezimal 16) zu ändern, und der Speicherbereich von $1300 (dezimal 4864) bis einschließlich $1529 (dezimal 5417) auf Diskette zu speichern. Auf diese Weise erhält man den reinen Druckertreiber ohne Hardcopy-Funktion.

Zum Anschluß eines Druckers mit Centronics-Schnittstelle wird ein Kabel mit Userport-Stecker und ein 36poliger Amphenol-Stecker benötigt. Als Kabel kann ein 11adriges Computer- oder ein 10adriges Fernmeldekabel mit Abschirmung verwendet werden. Die Länge des Kabels sollte 3 Meter nicht überschreiten. Die Kabel- beziehungsweise Pinbelegung können Sie Bild 2 entnehmen. (G.M. Ritter/nj)

80-Zeichen-Grafik

Das Grafik-Paket, das die 640 x 200-Punkte-Auflösung auf dem C128 unterstützt. Alle Grafikbefehle des Basic 7.0 stehen damit für den neuen HiRes-Modus zur Verfügung und können leicht programmiert werden.

Der neue Basic-Interpreter des Commodore 128 enthält 26 Befehle, mit deren Hilfe man einfach und schnell auch komplizierte Grafik-Programme erstellen kann. Diese Befehle wurden im Sonderheft 1/86 auf den 80-Zeichen-Bildschirm umgesetzt. Da im nachfolgenden Artikel auf »Graphik-80« zurückgegriffen wird, haben wir zum besseren Verständnis das Listing nochmal abgedruckt. Das abgedruckte Basic-Programm (Listing 1) baut ein Maschinenprogramm für den »User-RAM-Bereich« von $1300 bis $1bff auf. Nach dem Initialisieren dieses Maschinenprogramms mit »SYS DEC ("1303")« stehen die Befehle GRAFIK, BOX, CIRCLE, DRAW und PAINT auch für den VDC-Chip zur Verfügung. Die Befehle LOCATE, SCALE und SCNCLR und die Funktionen RCLR, RDOT und RGR können ohne Einschränkung im 640x200-Grafik-Modus benutzt werden. Beide Bildschirme können gleichzeitig im HiRes-Modus arbeiten, beim Aufruf eines Grafikbefehls prüft der Interpreter selbständig, welcher Video-Chip gerade angesprochen ist.

Die HiRes-Grafik-Register

Für die Grafikprogrammierung sind von den 36 Registern des VDC vor allem die Register 18, 19, 31, 25 und 26 interessant. Der Inhalt von Register 26 bestimmt (im Grafikmodus!) Vorder- und Hintergrundfarbe. Über Bit 0 bis 3 kann man 16 verschiedene Hintergrundfarben, über Bit 4 bis 7 16 verschiedene Vordergrundfarben anwählen.

Ein Beispiel: Die Befehlsfolge »POKE 54784,26:POKE 54785, (3*16+2)« bewirkt, daß im HiRes-Modus eine rote Zeichenfarbe auf weißem Grund erscheint. Im Textmodus wird mit diesem Befehl nur die Hintergrundfarbe verändert. Register 25 ist ein mehrfach belegtes Register. Über die Bits 0 bis 3 kann man den Bildschirm um maximal 16 Pixel horizontal nach links verschieben. Mit Bit 4 kann man zwischen einfacher (0) und doppelter (1) Pixelgröße wählen. Mit den Bits 5 bis 7 kann man zwischen den drei Betriebsarten Text (Bit 6=1), Semigrafik (Bit 5=1) und Grafik (Bit 7=1) wählen.

Für uns ist hier nur interessant, daß man durch Beschreiben von Register 25 mit dem Wert 128 den HiRes-Modus wählen und durch Beschreiben des Registers mit dem Wert 64 wieder in den Textmodus zurückkehren kann. Alle anderen Bits sind standardmäßig mit Null besetzt. Über die Register 18, 19 und 31 wird das Video-RAM angesprochen. Man besetzt in der beschriebenen Weise zunächst Register 18 mit dem High-Byte und Register 19 mit dem Low-Byte der gewünschten Bildschirmspeicheradresse.

Durch Schreiben in Register 31 kann man dann einen Wert in die gewählte Speicherstelle schreiben, durch Auslesen von Register 31 erhält man den Inhalt dieser Speicherstelle. Aber Vorsicht! Nach jeder Schreib- oder Leseoperation wird der RAM-Zeiger in Register 18/19 automatisch einen Schritt erhöht! Will man also einen Wert aus dem Video-RAM auslesen und anschließend (verändert) wieder zurückschreiben, so muß man die Zeiger 18/19 zweimal setzen. Das erscheint zwar sehr umständlich, doch hat diese Arbeitsweise einen überzeugenden Vorteil: 10mal »PEEK (Register 31)« ergibt die 10 Byte von Zeiger 18/19 bis Zeiger 18/19 + 9. Der Zeiger 18/19 muß nur einmal gesetzt werden. Das Auslesen oder Vorbesetzen größerer Speicherbereiche wird so erheblich beschleunigt.

Der neue GRAPHIC-Befehl

Der GRAPHIC-Befehl wurde um die Funktionen GRAPHIC 6,0 und GRAPHIC 6,1 erweitert. GRAPHIC 6 schaltet den 8563-HiRes-Modus ein. Folgt der »6« eine »1«, so wird der Bildschirm gelöscht, folgt eine »0«, so bleibt der Bildschirm, wie er ist. Der Befehl GRAPHIC 6,1 ersetzt auch den in dieser Implementation nicht vorgesehenen Befehl SCNCLR 6. Alle übrigen Informationen zur Implementation dieses Befehls sind im Sonderheft 1/86 als ausführlich kommentiertes Assemblerlisting abgedruckt.

Die »Originalroutine« im ROM befindet sich im Speicherbereich ab $6b5a.

Die Befehle BOX, CIRCLE, DRAW und PAINT

Diese Befehle funktionieren im 8563-Modus genauso, wie es im Bedienungshandbuch für die VIC-Grafik beschrieben ist. Einzige Änderung: Die x-Koordinaten dürfen nun im Bereich von 0 bis 639 liegen. Auch die Implementierung dieser Befehle ist denkbar einfach: Nacheinander werden die Programmteile PAINT ($61a8 bis $62b6), BOX ($62b7 bis $6388), DRAW ($6797 bis $67d6) und CIRCLE ($668e bis $674c) in den RAM-Bereich ab $1672 kopiert (Basic-Programm Zeile 5000 bis 5340). Darunter, in den RAM-Bereich ab $1952, wird der Programmteil »Strecke zeichnen« aus ROM $9b30 bis $9c18 kopiert. Als nächstes werden die neuen Adressen für Unterprogrammaufrufe eingesetzt (WHILE-DO-Schleife). Es folgt schließlich der neue GRAPHIC-Befehl. Im Basic-Programm ist er in Form von DATA-Zeilen abgelegt. Ebenfalls in Form von DATA-Zeilen sind die schon besprochene Routine »setpoint« (RAM $1400 bis $1671) und die Erweiterung der Interpreterschleife im Basic-Text enthalten.

Eingabehinweise: Geben Sie Graphik-80 ein und speichern Sie das Programm. Dann brauchen Sie nur noch als erste (!) Programmzeile »bload "graphik-80.m": sys dec ("1303")« einzugeben und das Grafik-Paket ist aktiviert. Alle Grafikbefehle beziehen sich jetzt auf den 80-Zeichen-Schirm. (Th. Rumbach/D. Winkler/ev/og)

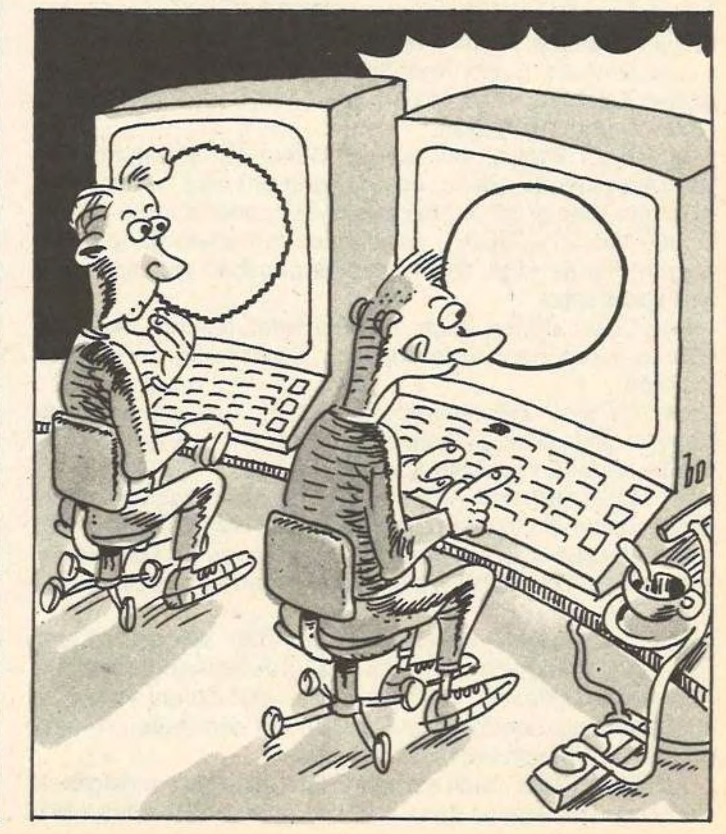
```
1 REM "{24SPACE}GRAPHIK-80
2 :
3 REM "{4SPACE}ERSTELLT GRAPHIK-PAKET FXR 80
  -ZEICHEN-HIRES-BILDSCHIRM
4 :
5 REM "{24SPACE}DES{2SPACE}C-128
8 :
9 REM "{23SPACE}VERSION 1.00
10 :
11 :
1000 FAST
1020 RESTORE
1040 BANK 15
1060 SCNCLR 5: PRINT CHR$(17) CHR$(17) TAB(2
     0)"ERSTELLEN EINES GRAPHIK-PAKETES"
```

Listing 1. »Grafik-80«, 640 x 200-Punkte-Auflösung

```
1080 PRINT CHR$(17) TAB(17)"FÜR DEN 8563-HIRES-GRAPHIK-BILDSCHIRM"
1100 PRINT CHR$(17) CHR$(17) CHR$(17) "{2SPACE}BEARBEITET:"
1120 PRINT : PRINT
1140 :
2000 REM "ERWEITERUNG DER INTERPRETERSCHLEIFE LADEN
2020 :
2040 PRINT TAB(25) "NEUE INTERPRETERSCHLEIFE
2060 A= DEC("1300"): E= DEC("1398")
2080 GOSUB 8000
2100 :
3000 REM "NEUE ROUTINE FÜR 'PUNKT SETZEN, LÖSCHEN UND TESTEN' LADEN
3020 :
3040 PRINT TAB(25) "PUNKT SETZEN, LÖSCHEN, TESTEN
3060 A= DEC("1400"): E= DEC("1671")
3080 GOSUB 8000
3100 :
4000 REM "ERWEITERUNG DER GRAPHIC-ROUTINE LADEN
4020 :
4040 PRINT TAB(25) "ERWEITERUNG DER GRAPHIK-ROUTINE
4060 A= DEC("1A3E"): E= DEC("1AB6")
4080 GOSUB 8000
4100 :
5000 REM "ROM-ROUTINEN IN RAM-BEREICH KOPIEREN
5020 :
5040 PRINT TAB(25) "ROM -> RAM - KOPIE
5060 A= DEC("61A8"): E= DEC("6388"): REM "PAINT UND BOX
5080 G= DEC("1672")
5100 GOSUB 8500
5120 :
5140 A= DEC("6797"): E= DEC("67D6"): REM "DRAW
5160 IF G<>DEC("1853") THEN PRINT "FEHLER !": STOP
5180 GOSUB 8500
5200 :
5220 A= DEC("668E"): E= DEC("674C"): REM "CIRCLE
5240 IF G<>DEC("1893") THEN PRINT "FEHLER !": STOP
5260 GOSUB 8500
5280 :
5300 A= DEC("9B30"): E= DEC("9C18"): REM "STRECKE ZEICHNEN
5320 IF G<>DEC("1952") THEN PRINT "FEHLER !": STOP
5340 GOSUB 8500
5360 :
5380 READ AD$
5400 DO WHILE AD$<>"ENDE"
5420 READ MN$, AL$, AH$
5440 AD= DEC(AD$)
5460 POKE AD, DEC(MN$): POKE AD+1, DEC(AL$): POKE AD+2, DEC(AH$)
5480 READ AD$
5500 LOOP
5520 :
5540 PRINT TAB(25) "FERTIGES MASCHINENPROGRAMM ABSPEICHERN
5560 IF DS >20 THEN PRINT DS$: STOP
5580 BSAVE "GRAPHIK-80.M",D0,U8,ON B0,P(DEC("1300")) TO P(DEC("1AB7"))
5600 IF DS >20 THEN PRINT DS$: STOP
5620 SYS DEC("1303")
5640 PRINT CHR$(17) CHR$(17) "{5SPACE}GRAPHIK-PAKET FERTIG INSTALLIERT, ";
5645 PRINT "GESPEICHERT UND INITIALISIERT."
5660 END
6000 END
7960 REM "KOPIERROUTINE 1
7980 :
8000 FOR I=A TO E
8020 READ D
8040 POKE I,D
8060 NEXT I
8080 RETURN
8100 :
8460 REM "KOPIERROUTINE 2
8480 :
8500 FOR I=A TO E
8520 POKE G, PEEK(I)
8540 G= G+1
8560 NEXT I
8580 RETURN
8600 :
9000 :
10000 REM "DATAS FÜR NEUE INTERPRETERSCHLEIFE
10010 :
10020 DATA  76,45,19,76,6,19,120,169,45,141,8,3,169,19,141,9,3,88,169,0,141,0
10030 :
10040 DATA 255,169,6,141,6,213,169,32,133,48,133,50,133,52,169,0,133,47,133
10050 :
10060 DATA  49,133,51,96,32,128,3,201,222,144,31,201,233,176,27,170,36,215,16
10070 :
10080 DATA  52,189,123,18,141,78,19,189,135,18,141,79,19,138,32,128,3,32,0,0
10090 :
10100 DATA  76,246,74,32,134,3,76,243,74,62,114,215,129,147,141,43,86,88,226
10110 :
10120 DATA 121,96,26,22,103,23,24,101,100,24,105,105,106,105,189,162,18,141
10130 :
10140 DATA  78,19,189,174,18,141,79,19,76,73,19,90,168,215,183,142,141,43,151
10150 :
10160 DATA  85,226,121,96,107,97,103,98,102,101,100,103,105,105,106,105,85
10170 :
12000 REM "DATAS FÜR ROUTINE PUNKT SETZEN, LÖSCHEN, TESTEN
12010 :
12020 DATA  76,48,22,76,9,20,76,93,22,173,0,255,72,169,0,141,0,255,169,7,141
12030 :
12040 DATA  6,213,104,141,0,255,96,173,0,255,133,158,169,0,141,0,255,133,253
12050 :
12060 DATA 133,254,173,52,17,208,90,169,199,205,51,17,144,83,169,127,237,49
12070 :
12080 DATA 17,169,2,237,50,17,144,71,172,51,17,185,150,20,133,253,185,99,21
12090 :
12100 DATA 133,254,174,50,17,173,49,17,41,248,74,74,74,24,125,147,20,133,252
12110 :
12120 DATA 24,165,253,101,252,133,253,165,254,105,0,133,254,162,18,32,204,205
12130 :
12140 DATA 162,19,165,253,32,204,205,32,216,205,133,252,173,49,17,41,7,170,189
12150 :
12160 DATA 139,20,24,96,56,96,128,64,32,16,8,4,2,1,0,32,64,0,80,160,240,64,144
12170 :
12180 DATA 224,48,128,208,32,112,192,16,96,176,0,80,160,240,64,144,224,48,128
12190 :
12200 DATA 208,32,112,192,16,96,176,0,80,160,240,64,144,224,48,128,208,32,112
12210 :
12220 DATA 192,16,96,176,0,80,160,240,64,144,224,48,128,208,32,112,192,16,96
12230 :
12240 DATA 176,0,80,160,240,64,144,224,48,12
```

```
         8,208,32,112,192,16,96,176,0,80
12250 :
12260 DATA 160,240,64,144,224,48,128,208,32,
         112,192,16,96,176,0,80,160,240,64
12270 :
12280 DATA 144,224,48,128,208,32,112,192,16,
         96,176,0,80,160,240,64,144,224,48
12290 :
12300 DATA 128,208,32,112,192,16,96,176,0,80
         ,160,240,64,144,224,48,128,208,32
12310 :
12320 DATA 112,192,16,96,176,0,80,160,240,64
         ,144,224,48,128,208,32,112,192,16
12330 :
12340 DATA 96,176,0,80,160,240,64,144,224,48
         ,128,208,32,112,192,16,96,176,0
12350 :
12360 DATA 80,160,240,64,144,224,48,128,208,
         32,112,192,16,96,176,0,80,160,240
12370 :
12380 DATA 64,144,224,48,128,208,32,112,192,
         0,0,0,0,1,1,1,2,2,2,3,3,3,4,4,4
12390 :
12400 DATA 5,5,5,5,6,6,6,7,7,7,8,8,8,9,9,9,1
         0,10,10,10,11,11,11,12,12,12,13
12410 :
12420 DATA 13,13,14,14,14,15,15,15,15,16,16,
         16,17,17,17,18,18,18,19,19,19,20
12430 :
12440 DATA 20,20,20,21,21,21,22,22,22,23,23,
         23,24,24,24,25,25,25,25,26,26,26
12450 :
12460 DATA 27,27,27,28,28,28,29,29,29,30,30,
         30,30,31,31,31,32,32,32,33,33,33
12470 :
12480 DATA 34,34,34,35,35,35,35,36,36,36,37,
         37,37,38,38,38,39,39,39,40,40,40
12490 :
12500 DATA 40,41,41,41,42,42,42,43,43,43,44,
         44,44,45,45,45,45,46,46,46,47,47
12510 :
12520 DATA 47,48,48,48,49,49,49,50,50,50,50,
         51,51,51,52,52,52,53,53,53,54,54
12530 :
12540 DATA 54,55,55,55,55,56,56,56,57,57,57,
         58,58,58,59,59,59,60,60,60,60,61
12550 :
12560 DATA 61,61,62,62,62,63,63,63,32,28,20,
         176,34,166,131,208,5,73,255,37,252
12570 :
12580 DATA  44,5,252,133,252,162,18,165,254,
         32,204,205,162,19,165,253,32,204
12590 :
12600 DATA 205,162,31,165,252,32,204,205,165
         ,158,141,0,255,96,32,28,20,176,245
12610 :
12620 DATA 37,252,240,6,32,87,22,162,0,96,32
         ,87,22,162,255,96
12630 :
14000 REM "DATAS FXR ERWEITERUNG DES GRAPHIC
         -BEFEHLS
14010 :
14020 DATA 201,158,208,11,32,34,160,32,128,3
         ,169,0,133,216,96,32,244,135,224
14030 :
14040 DATA 6,240,41,176,36,138,72,169,0,141,
         0,255,162,25,169,64,32,204,205,32
14050 :
14060 DATA 12,206,165,215,72,169,128,133,215
         ,32,66,193,104,133,215,104,72,170
14070 :
14080 DATA  76,110,107,76,40,125,169,0,141,0
         ,255,162,25,169,128,32,204,205,32
14090 :
14100 DATA 28,158,169,0,141,0,255,224,2,176,
         229,224,0,240,29,160,64,132,8,160
14110 :
14120 DATA 0,152,162,18,32,204,205,162,19,32
         ,204,205,162,31,32,204,205,136,208
14130 :
14140 DATA 250,198,8,208,246,96
19900 :
20000 DATA 1A3B, 4C, 00, 14
20010 DATA 16D5, 20, 00, 14
20020 DATA 1A2D, 20, 00, 14
20030 DATA 188D, 20, 1D, 1A
20040 DATA 19C7, 20, 1D, 1A
20050 DATA 19EB, 20, 0C, 1A
20060 DATA 1A07, 20, 0C, 1A
20070 DATA 1887, 4C, 6C, 18
20080 DATA 1890, 4C, 6C, 18
20090 DATA 17B1, 20, 52, 19
20100 DATA 17F1, 20, 52, 19
20110 DATA 1884, 20, 52, 19
20120 DATA 1948, 20, 52, 19
20130 DATA 1692, 20, 06, 14
20140 DATA 16C6, 20, 06, 14
20150 DATA 1712, 20, 06, 14
20160 DATA 1747, 20, 06, 14
20170 DATA 16E5, 20, 46, 17
20180 DATA 16FA, 20, 46, 17
20190 DATA 1740, 4C, B5, 16
20200 DATA 1869, 4C, 1D, 1A
20210 DATA 1672, 20, 32, 9E
20220 DATA 1781, 20, 32, 9E
20230 DATA 1893, 20, 32, 9E
20240 DATA 1853, EA, EA, EA
20250 DATA ENDE
50000 :
60000 ZZ$="GRAPHIK-80": UN=8
60010 OPEN 15,UN,15,"S0:"+ZZ$
60020 GOSUB 60100
60030 SAVE ZZ$,UN
60040 GOSUB 60100
60050 VERIFY ZZ$,UN
60060 CLOSE 15
60070 END
60100 INPUT#15, S1, S$, S2, S3
60110 IF S1=1 THEN PRINT S2; S$
60120 IF S1<20 THEN RETURN
60130 PRINT S1 ", " S$ "," S2; "," S3
60140 CLOSE 15
```

Listing 1. »Grafik-80« eine Erweiterung, die die 640 x 200-Punkte-Auflösung unterstützt (Schluß)



Character 80

Die erste Erweiterung zu Graphik 80 ist die des CHAR-Befehls. Er beherrscht jetzt verschiedene Schriftgrößen in mehreren Richtungen und ist variabler als der PRINT-Befehl.

Das im folgenden Text beschriebene Programm »Character 80« ist eine Ergänzung des 640x200-Punkte-Grafikpaketes »Graphik 80« für den C 128, das in Ausgabe 12/85 vorgestellt und auf den vorhergehenden Seiten noch einmal abgedruckt wurde. »Character 80« fügt in den Basic-Interpreter des C 128 einen neuen CHAR-Befehl ein, der bei eingeschaltetem VDC-Grafikbildschirm wirksam wird. Der CHAR-Befehl dient dazu, Buchstaben, Ziffern oder ganze Textzeilen auf dem Grafikbildschirm darzustellen. Er vertritt den PRINT-Befehl, den man nur im Textmodus verwenden kann. Man benötigt den CHAR-Befehl zum Beispiel, um die Achsen eines Koordinatensystems zu beschriften oder um den Spielstand in einem Spiel mit Grafikdarstellung einzublenden. Ein wichtiger Befehl also, der nun auch in Graphik 80 nicht länger fehlen soll.

Der neue CHAR-Befehl

Die CHAR-Routine für den 80-Zeichen-Bildschirm wurde völlig neu geschrieben und gegenüber der im Basic 7.0 vorgesehenen CHAR-Routine für den 40-Zeichen-Bildschirm stark erweitert. Aus diesem Grund hat sich die Syntax des Befehls geändert. Der neue Befehl lautet:

```
CHAR [f],x,y,[w],q$
```

f: ist ein ganzzahliger Wert zwischen 0 und 1. Wird für f eine Null angegeben, so wird der Text in q$ in der Farbe des Bildschirmhintergrunds ausgegeben. Nimmt f den Wert eins an, dann wird der Text in q$ in der Zeichenfarbe ausgegeben (vergleiche Beschreibung des COLOR-Befehls im Bedienungshandbuch, Seite 4 bis 35).

x,y: sind die Grafik-Koordinaten des Mittelpunktes des ersten Zeichens in q$. x darf zwischen 0 und 639, y zwischen 0 und 199 liegen.

w: ist der Winkel, um den der String q$ gegenüber der Waagerechten (x-Achse) um (x,y) gedreht wird. w darf zwar eine beliebige positive Integerzahl sein, aber nur die Winkel 0, 90, 180, 270, 360, ... Grad ergeben lesbare Textdarstellungen. Für den Fall, daß w nicht angegeben wird, ist w mit Null vorbesetzt.

Für f,x,y,w dürfen auch Variable oder numerische Ausdrücke, deren Ergebnis im erlaubten Bereich liegt, eingesetzt werden.

q$: ist eine Zeichenkette in Anführungszeichen, eine Stringvariable oder ein beliebiger Ausdruck des Typs String. Steuerzeichen innerhalb von »q$« bis auf ⟨RVS ON⟩, ⟨RVS OFF⟩, ⟨CSR UP⟩ und ⟨CSR DOWN⟩ werden ignoriert. ⟨RVS ON⟩ schaltet die Invers-Darstellung ein, ⟨RVS OFF⟩ schaltet sie wieder aus. ⟨CSR UP⟩ setzt den folgenden Text eine Halbzeile höher, ⟨CSR DOWN⟩ setzt den folgenden Text eine Halbzeile tiefer.

Der neue CHAR-Befehl wird nur bei eingeschalteter 640x200-Punkte-Grafik aktiv. Im 80-Zeichen-Textmodus oder bei VIC-Grafiken bleibt der alte CHAR-Befehl so, wie er im Bedienungshandbuch des C 128 auf den Seiten 4 - 28 und 4 - 29 beschrieben ist, erhalten.

An dieser Stelle seien ein paar Sätze erlaubt zum Vergleich der Basic 7.0-Version des CHAR-Befehls und Character 80. Nach der alten Version des CHAR-Befehls durften x und y nur Werte zwischen 0 und 79 beziehungsweise 0 und 24 annehmen. Auf dem Grafikbildschirm wurde eine Textdarstellung simuliert, wie sie auch im Textmodus möglich ist. Diese Methode hat den Vorteil, daß der CHAR-Befehl auch im Textmodus zur direkten Positionierung einer Textzeile angewandt werden kann. Punktgenaues Einfügen eines Textes in ein Grafikbild – etwa bei der Beschriftung von Funktionsgraphen – ist jedoch nur möglich, wenn die Position des Grafik-Cursors als Bezugspunkt für den Beginn der Zeichenkette dient. Character 80 betrachtet die in x und y angegebene Grafik-Cursorposition als Mittelpunkt des ersten Buchstabens in der Zeichenkette q$.

Der nächste und vielleicht auffälligste Unterschied zwischen neuer und alter CHAR-Routine ist die Möglichkeit, Textzeilen nicht nur waagerecht (w=0), sondern auch senkrecht aufwärts (w=90) und abwärts (w=270) und auf dem Kopf stehend (w=180) auszugeben. Dabei wird berücksichtigt, daß die Auflösung in x-Richtung mit 640 Punkten wesentlich höher ist als die 200-Punkte-Auflösung in y-Richtung. Durch Einführen eines drehwinkelabhängigen Korrekturfaktors wurde erreicht, daß ein und dieselbe Textzeile in y-Richtung fast genauso lang und so hoch erscheint wie in x-Richtung. Wie oben dargestellt, werden für w auch andere als die genannten Werte akzeptiert. Die Auflösung von 8 x 8 Punkten pro Buchstaben reicht aber leider nicht, um für beliebige Winkel lesbare Darstellungen zu erreichen. Character 80 kann auf zwei verschiedene Schriftgrößen eingestellt werden. Voreingestellt ist eine vergrößerte Schrift, die etwa 40 Zeichen in der Waagerechten erlaubt. Mit

```
POKE 5061,128: POKE 5066,144: POKE 5108,48
```

kann auch auf die normale 80-Zeichen-Schrift umgestellt werden. Mit

```
POKE 5061,223: POKE 5066,239: POKE 5108,192
```

kann die vergrößerte Schrift gewählt werden. Vor allem beim Senkrechtschreiben ist die zuletzt genannte Betriebsart der besseren Lesbarkeit wegen vorzuziehen.

Und noch eine dritte wichtige Änderung der CHAR-Routine wurde mit Character 80 eingeführt: Abhängig davon, ob im Textmodus (mit CHR$(14) beziehungsweise CHR$(142)) Groß-/Grafik- oder Groß-/Kleinschrift eingestellt wurde, stellt auch der neue CHAR-Befehl Groß-/Grafik- oder Groß-/Kleinschrift dar. Dadurch wird der gesamte Zeichensatz des CHAR-ROMs nutzbar. Die Umschaltung Invers-/Normalschrift kann jetzt innerhalb von q$ mit den vom PRINT-Befehl her bekannten Steuerzeichen vorgenommen werden, ebenso wie das Hoch- und Tiefstellen von ganzen Textteilen in Halbschritten. Auf diese Weise kann eine ganze Textzeile, egal wie viele Indizes, Potenzen und Inversteile sie enthält, mit einem einzigen Befehl ausgegeben werden. Auch mehrfaches Tiefer- oder Höherstellen hintereinander wird erkannt.

Nach der ausführlichen Beschreibung der Möglichkeiten der neuen CHAR-Routine nun zum praktischen Teil! Es folgt ein »Sechs-Stufen-Plan« zum Einfügen des neuen Befehls in das Grafikpaket Graphik 80.

1) Laden Sie das Basic-Programm Graphik 80.
2) Fügen Sie die abgedruckten Basic-Zeilen (Listing 1) in das Programm ein.
3) Ersetzen Sie die Zahl 215 in Zeile 10100 durch die Zahl 183 und die Zahl 103 in Zeile 10120 durch die Zahl 26.
4) Ersetzen Sie in Zeile 5580 die Endadresse des zu speichernden Bereichs ($1ab7) durch die neue Endadresse ($1c00).

5) Löschen Sie die alte und speichern Sie die neue Version von Graphik 80.
6) Starten Sie das Programm mit RUN.

Wenn das Programm ohne Fehlermeldung abgelaufen ist und sich kein Fehler beim Abtippen der DATA-Zeilen eingeschlichen hat, dann befindet sich jetzt das fertige neue Grafikpaket Graphik 80 auf Ihrer Diskette. Sie können es wie gewohnt mit

```
BLOAD "GRAPHIK-80.m": SYS DEC ("1303")
```

laden und starten.

Ein bißchen Theorie

Die meisten Leser, vorausgesetzt, daß sie bis hierher durchgehalten haben, werden nun fragen: Wie funktioniert der CHAR-Befehl überhaupt? Auch Leser, die (noch) keinen C 128 besitzen und ihren eigenen CHAR-Befehl schreiben wollen, werden sich dafür interessieren, nach welchem Prinzip der CHAR-Befehl aufgebaut ist:

Allen Zeichen des CBM-Zeichensatzes liegt eine Matrix aus 8 x 8 Punkten zugrunde. Acht nebeneinanderliegende Punkte in der Horizontalen werden von den acht Bit eines Bytes im CHAR-ROM dargestellt. Acht solcher Zeilen (entsprechend acht Byte im CHAR-ROM) in der Vertikalen bilden die Matrix für ein Zeichen. Alle Bits in dieser Matrix, die den Wert »1« haben, liefern einen farbigen Punkt auf dem Bildschirm. Alle Bits mit dem Wert »0« werden in der Farbe des Bildschirmhintergrundes dargestellt. Es gibt 128 verschiedene Zeichen im Commodore-Zeichensatz, die wahlweise normal oder invertiert ausgegeben werden können. Insgesamt werden also 256 x 8 = 2 KByte Speicherplatz für einen Zeichensatz benötigt. Welches dieser 256 verschiedenen Zeichen an welcher Stelle auf dem Bildschirm erscheinen soll, das ist im Bildschirmspeicherbereich vermerkt: Jedem Platz auf dem Bildschirm ist genau ein Platz im Video-RAM zugeordnet. Steht im zehnten Speicherplatz des Video-RAMs eine »24«, dann wird an der zehnten Position auf dem Bildschirm das 24. Zeichen des CHAR-ROMs dargestellt. Im CHAR-ROM sind für dieses Zeichen die 8 Byte von 24 x 8 (=192) bis 24 x 8 + 7 (=199) reserviert. Zusätzlich zum Video-RAM gibt es noch einen genauso großen Farbspeicherbereich. Zu jedem Byte im Video-RAM gehört genau ein Byte im Color-RAM. In diesem Byte ist verzeichnet, in welcher Farbe das Zeichen erscheinen soll, und – im Falle des 8563 im C 128 – ob es blinkend oder mit Unterstreichung dargestellt wird. Diese Überlegungen gelten nur für den Textmodus. In dieser Betriebsart übernimmt der Videocontroller die Aufgabe, die den Zeichen des Video-RAM entsprechenden Bytes aus dem CHAR-ROM zu holen und an der richtigen Stelle auf den Bildschirm zu zeichnen. Schaltet man dagegen den Grafikmodus ein, dann wird die gesamte Zeicheninformation nur aus dem Video-RAM ausgelesen, der CHAR-ROM-Bereich wird nicht mehr angesprochen. Gleichzeitig wird aber ein sehr viel größerer Bildschirmspeicher benötigt: 1 Bit für jeden Punkt, also rund 16 KByte bei einer Auflösung von 200 x 640 Punkten (mehr zu diesem Thema im Beitrag Graphik 80). Da aber der Videocontroller nur Punkte darstellen kann, braucht man ein eigenes Programm, um die Zeichen aus dem CHAR-ROM auf dem Bildschirm abzubilden. Alle 64 Punkte der 8 x 8-Matrix müssen einzeln ermittelt und über die Routine »Punkt setzen« ausgegeben werden. Genauso macht es auch Character 80. Zunächst wird der Grafik-Cursor vom Mittelpunkt auf den rechten oberen Eckpunkt der 8 x 8-Matrix verschoben. Dann wird das erste Byte des Buchstabens, den man darstellen will, aus dem CHAR-ROM geholt. Man liest das erste Bit dieses Bytes und setzt einen Punkt an die Position des Grafik-Cursors, wenn das Bit »1« ist. Ist das Bit »0«, dann geschieht nichts. Danach wird der Cursor in waagerechter Richtung um einen Punkt nach links verschoben. Man liest das zweite Bit, gibt den nächsten Punkt aus, rückt den Cursor eins weiter nach links und so weiter, bis alle acht Bit abgearbeitet sind. Dann holt man das zweite Byte des gewählten Buchstabens, setzt den Cursor eins tiefer und anschließend auf den rechten Rand zurück. Von dort beginnt das Spiel von vorn: Achtmal untereinander, acht Punkte nebeneinander. Dabei ist es unwesentlich, ob man, wie hier für Character 80 beschrieben, rechts oben oder links unten anfängt. Nachdem der letzte Punkt des Buchstabens gezeichnet ist, wird der Grafik-Cursor auf den Mittelpunkt des folgenden Buchstabens verschoben, die Routine ist bereit für das nächste Zeichen.

Welche Zeichen ausgegeben werden, das steht in der Zeichenkette q$. CHAR geht so vor: Es nimmt den ersten Buchstaben der Kette, stellt fest, das wievielte Zeichen im CHAR-ROM diesem Buchstaben entspricht (CBM-ASCII – Bildschirmcode – Wandlung) und gibt dieses Zeichen aus. Dann wird das zweite Zeichen geholt, ausgewertet und ausgegeben und so weiter bis zum Ende der Zeichenkette. Steuerzeichen innerhalb dieses Strings werden einfach übergangen. Nur die Steuerzeichen ⟨CSR UP⟩, ⟨CSR DOWN⟩, ⟨RVS ON⟩ und ⟨RVS OFF⟩ werden (in unserem Fall!) erkannt. ⟨CSR UP⟩ bewirkt, daß der Grafik-Cursor vier Zeilen (1/2 Buchstabe) nach oben verschoben wird, ⟨CSR DOWN⟩schiebt ihn vier Zeilen nach unten. ⟨RVS ON⟩ setzt ein Flag, das bei der Ermittlung der CHAR-ROM-Adresse abgefragt wird. ⟨RVS OFF⟩ löscht dieses Flag wieder. Das RVS ON-Flag ist die Vier, das RVS OFF-Flag die Null. Die Zahlen wurden gerade deshalb ausgewählt, weil man so nur das RVS-Flag zum High-Byte der CHAR-ROM-Anfangsadresse addieren muß, um den richtigen Speicherbereich zu treffen. Ein Beispiel zur Erklärung: Der CHAR-ROM-Speicher für Groß-/Kleinschrift beginnt bei $d800, der inverse Groß-/Kleinschriftzeichensatz bei $dc00; ist der Inversmodus ausgeschaltet, dann ist das RVS-Flag 0: $d8+$0=$d8; bei eingeschaltetem Inversmodus ist das RVS-Flag 4: $d8+$4=$dc. Ob als Anfangsadresse $d000 für Groß-/Grafikschrift oder $d800 für Groß-/Kleinschrift gewählt wird, das hängt vom Inhalt der Speicherstelle $f1 ab, die unter anderem anzeigt, welcher der beiden Zeichensätze schon im Textmodus voreingestellt wurde.

Koordinatentransformation

Nun fehlt nur noch ein Punkt in der Beschreibung des Prinzips, nach dem Character 80 arbeitet: Wie dreht man die dargestellte Zeichenkette um den Mittelpunkt des ersten Buchstabens?

Der Mittelpunkt (x,y) des ersten Buchstabens ist der Ursprung eines gedachten kartesischen Koordinatensystems. Die Horizontale des Bildschirms ist die Richtung der x-Achse dieses Koordinatensystems, die Vertikale des Bildschirms ist die Richtung der y-Achse. Alle Bildpunkte innerhalb einer zusammenhängenden Zeichenkette eines CHAR-Befehls werden relativ zum Ursprungspunkt dieses Koordinatensystems berechnet: Der Mittelpunkt des zweiten Buchstabens erhält also zum Beispiel die Koordinaten (0,8), die rechte obere Kante des dritten Buchstabens die Koordinaten (−4,20) (y-Richtung wie die physikalische Richtung von oben nach unten). Alle Punkte eines Strings werden nach diesem Schema berechnet. Anschließend wird das Koordinatensystem um den Winkel w gedreht. Man könnte ebenso sagen: Der berechnete Punkt wird um den Ursprung des Koordinatensystems gedreht; der Algorithmus bleibt der gleiche. Er lautet:

$x_{neu} = w + y \cdot \sin w$

$y_{neu} = y \cdot \cos w - x \cdot \sin w$

Darin sind x und y die nicht gedrehten Koordinaten (relativ

```
4200 REM " *** NEUE CHAR-ROUTINE LADEN *** "
4220 :
4240 PRINT TAB(25) "NEUE CHAR-ROUTINE"
4260 A= DEC("1AB7"): E= DEC("1BFF")
4280 GOSUB 8000
4300 :
4320 A= DEC("139E"): E= DEC("13FF")
4340 GOSUB 8000
4360 :
5550 SCRATCH "GRAPHIK-80.M"
14150 :
15000 REM " *** DATAS FÜR DIE CHAR-ROUTINE *
      ** "
15010 :
15020 DATA 169,0,141,0,255,162,25,32,218,205
      ,48,3,76,215,103,32,50,158,162,31
15030 :
15040 DATA  32,82,158,32,6,158,140,92,17,141
      ,93,17,169,0,162,7,157,96,17,202
15050 :
15060 DATA  16,250,133,113,32,92,121,32,123,
      135,133,112,164,113,196,112,240,19
15070 :
15080 DATA 169,127,141,0,255,177,36,162,0,14
      2,0,255,32,5,27,230,113,208,231
15090 :
15100 DATA  96,168,201,18,208,4,169,4,208,6,
      201,146,208,5,169,0,133,243,96,201
15110 :
15120 DATA 145,208,15,56,173,98,17,233,4,141
      ,98,17,176,3,206,99,17,96,201,17
15130 :
15140 DATA 208,15,24,173,98,17,105,4,141,98,
      17,144,3,238,99,17,96,41,127,201
15150 :
15160 DATA  32,144,193,152,201,128,176,12,20
      1,96,144,4,41,223,176,8,41,191,144
15170 :
15180 DATA   4,41,127,9,64,162,0,160,3,133,1
      08,134,109,6,108,38,109,136,208,249
15190 :
15200 DATA  24,165,108,105,0,133,108,165,109
      ,166,241,48,3,105,208,44,105,216
15210 :
15220 DATA 101,243,133,109,32,239,27,162,1,1
      42,0,255,160,7,177,108,72,136,16
15230 :
15240 DATA 250,162,0,142,0,255,162,8,134,106
      ,134,107,104,42,72,144,6,32,158
15250 :
15260 DATA  19,32,0,20,238,96,17,208,3,238,9
      7,17,198,107,208,233,24,169,1,109
15270 :
15280 DATA  98,17,141,98,17,144,3,238,99,17,
      104,56,173,96,17,233,8,141,96,17
15290 :
15300 DATA 176,3,206,97,17,162,8,198,106,208
      ,196,56,173,98,17,233,4,141,98,17
15310 :
15320 DATA 176,3,206,99,17,24,173,96,17,105,
      12,141,96,17,144,3,238,97,17,96
15330 :
15340 DATA  56,173,96,17,233,4,141,96,17,176
      ,3,206,97,17,76,27,27
15350 :
15360 DATA 160,43,32,116,154,162,3,189,96,17
      ,157,84,17,157,88,17,202,16,244
15370 :
15380 DATA 169,144,32,243,154,14,84,17,46,85
      ,17,14,86,17,46,87,17,162,39,160
15390 :
15400 DATA 223,32,243,19,160,239,162,35,32,2
      43,19,160,37,32,109,157,162,31,32
15410 :
15420 DATA 112,157,141,49,17,140,50,17,162,3
      9,160,41,32,124,157,162,33,32,112
15430 :
15440 DATA 157,141,51,17,140,52,17,96,169,19
      2,32,174,157,157,49,17,152,157,50
15450 :
15460 DATA 17,96
```

Listing 1. Fügen Sie die Basic-Zeilen in das Basic-Programm »Graphik 80« ein und nehmen Sie bitte die im Text erwähnten Änderungen vor

zum Ursprung des gedachten Koordinatensystems) und x_{neu} und y_{neu} die Koordinaten des gedrehten Koordinatensystems, die tatsächlich auf dem Bildschirm erscheinen. Nun taucht aber noch ein kleines Problem auf:

In x-Richtung ist die Auflösung unseres Bildschirms wesentlich höher als in y-Richtung, man sieht auch im HiRes-Modus die Bildschirmzeilen, aber nicht die -spalten. Deshalb müssen noch Korrekturfaktoren eingeführt werden, die dafür sorgen, daß dasselbe Wort in y-Richtung genauso lang erscheint wie in x-Richtung. Die vollständige Formel lautet dann:

$x_{neu} = k_1 \cdot x \cdot \cos w + k_2 \cdot y \cdot \sin w$

$y_{neu} = y \cdot \cos w - k_3 \cdot x \cdot \sin w$

Als günstigste Werte für die Korrekturfaktoren wurden gefunden:

$k_1 = 1{,}873$; $k_2 = 2$; $k_3 = 0{,}874$.

Damit erhält man eine im Vergleich zum Textmodus leicht vergrößerte Darstellung. Mit $k_1 = 1{,}126$; $k_2 = 2$ und $k_3 = 0{,}501$ erhält man die Schrift in Originalgröße. In der Praxis (in unserem Fall) wird so verfahren, daß zuerst die einfachen Koordinaten ohne Drehung berechnet werden, dann die Drehung ausgeführt (im Assemblerlisting »jsr rotate«) und schließlich wird der berechnete Punkt eingezeichnet. Der beschriebene Algorithmus für die Drehung des Koordinatensystems um den Ursprung – zumindest in der allgemeinen Form ohne Korrekturfaktoren – ist übrigens nicht nur für Zeichendarstellung geeignet, sondern ein allgemeingültiger Koordinatentransformations-Algorithmus. Wer ihn für eigene Programme benutzen will, der findet im Assemblerlisting zu Character 80 das ohne die anderen Programmteile von Graphik 80 lauffähige Unterprogramm »rotate«. Dort ist die Parameterübergabe an das Unterprogramm genau beschrieben. Im Assemblerlisting Character 80 ist auch die praktische Programmierung der gesamten CHAR-Anweisung beschrieben, weitere Kommentare sind also wohl nicht mehr nötig.

Zum Schluß noch ein Tip für experimentierfreudige Assemblerprogrammierer: Das Programm ist so geschrieben, daß Drehungen der Schrift um beliebige Winkel möglich sind. Leider sind die Schriftzeichen bei einer Auflösung von 8 x 8 Punkten bei nicht-rechtwinkligen Drehungen unleserlich. Abhilfe könnte hier vielleicht eine Verdopplung der Auflösung auf 16 x 16 Punkte schaffen. Im Assemblerlisting müssen zu diesem Zweck nur wenige Konstanten geändert werden. Die meiste Arbeit ist es aber, einen neuen Zeichensatz mit 32 Byte pro Zeichen zu entwickeln. Doch nur Mut, wir sind gespannt auf Ihre Ergebnisse. Und noch ein Tip: Der Basic-Interpreter akzeptiert bei manchen Befehlen außer »absoluten«, kartesischen Koordinaten (x,y) auch Polarkoordinaten der Form (r;w). Der Interpreter erkennt Polarkoordinaten daran, daß x und y nicht durch ein Komma, sondern durch ein Semikolon voneinander getrennt sind. Der Radius r ist die Zahl vor dem Semikolon, der Winkel w steht nach dem Semikolon. Die Verfasser haben das folgende Beispiel ausprobiert:

```
10 GRAPHIC 6,1
20 FOR I=0 TO 360
30 DRAW ,320,100 TO 80;I
40 DRAW 0,320,100 TO 80;I
50 NEXT I
60 GRAPHIC 5
```

(Dieter Winkler/Thomas Rumbach/og)

Hardcopy 80

Dieses Programm erweitert »Graphik-80« um die wichtige Möglichkeit, von dem 80-Zeichen-HiRes-Bildschirm des C128 Hardcopies in verschiedenen Größen auf einem Epson FX-80 oder kompatiblen Drucker ausgeben zu können. Der Drucker kann über den seriellen Bus oder über die in dieses Programm eingebaute Centronics-Schnittstelle angesprochen werden.

Wie mittlerweile allgemein bekannt ist, kann der C128 auch auf dem 80-Zeichen-Bildschirm hochauflösende Grafik darstellen. Die Auflösung von 640 x 200 Punkten (Pixel) ist geradezu eine Aufforderung, mit dieser im wahrsten Sinne des Wortes hochauflösenden Grafik zu arbeiten. Mit dem Programmpaket GRAPHIK-80 wurde es möglich, den 640 x 200 Punkte HiRes-Bildschirm des C128 von Basic aus anzusprechen. Trotz der mächtigen Befehle, die damit zur Verfügung stehen, fehlt ein Befehl, der es einem ermöglicht, seine Kunstwerke zu drucken. Diesem Mißstand helfen wir mit dieser Programmerweiterung ab. Das Ergebnis unserer Arbeit ist ein Programm (Listing 1), das standardmäßig mit den folgenden Druckmodi aufwarten kann:

1. Ausdruck in einfacher Größe
2. Ausdruck in einfacher Größe (revers)
3. Ausdruck in doppelter Größe
4. Ausdruck in doppelter Größe (revers)

Andere Bildgrößen und Punktdichten kann sich jeder individuell in Basic mit Hilfe des Installationsteils des Basic-Laders zusammenbasteln. Selbstverständlich ist die Druckausgabe mit jeder benötigten Sekundäradresse, mit oder ohne Line-Feed (LF) oder auf Wunsch über die eingebaute Centronics-Schnittstelle möglich.

Der maschinennahe Basic- und Assembler-Programmierer sieht beim Überfliegen der Assembler-Routinen (Listing 2) sehr schnell, daß durch Zufügen von Steuerbytes weit mehr als 4 Bildformate (bis 255) vorgewählt werden können. Das Programm ist in der Lage, HiRes-Bilder beim Drucken in x- und y-Richtung bis zu dem Faktor 3 beziehungsweise 4 zu vergrößern. Dies ergibt im Extremfall ein Bild mit über 1,5 Millionen Pixel auf dem Drucker, die dadurch zustande kommen, daß das Bild nicht einfach gedehnt wird, sondern bei gleicher Auflösung wie auf dem Bildschirm werden durch geschicktes Vervielfachen der einzelnen Pixel mehr Punkte ausgegeben.

Vor dem eigentlichen Programmstart muß natürlich das Programm in den Computer eingetippt werden. Speichern Sie das Programm danach. Somit haben Sie immer eine Ur-Version, die Sie mit Hilfe des Installationsteils Ihren jeweiligen Bedürfnissen anpassen können.

Starten Sie jetzt das Programm mit RUN. Es erzeugt zuerst ein Maschinenprogramm im Speicher und arbeitet anschließend den Installationsteil ab. Für diesen Teil benötigt das Programm von Ihnen verschiedene Informationen, die es im Dialog anfordert.

Wenn Sie nicht mit der eingebauten Centronics-Schnittstelle arbeiten, sollten Sie der Wahl der Sekundäradresse besondere Bedeutung zukommen lassen. Eine falsche Angabe führt im allgemeinen zu »Buchstabensalat mit ASCII-Allerlei«. Da dieses »Menü« nur in den seltensten Fällen Liebhaber findet, setzen Sie bitte die Sekundäradresse ein, bei der Ihr Interface keine Umkodierung der Zeichen (von Commodore nach ASCII) vornimmt. Diese Adresse steht in dem Handbuch, das Ihnen mit dem Interface mitgeliefert wurde. Anderenfalls müssen Sie etwas experimentieren. Dazu ein Tip: häufig gebrauchte Sekundäradressen sind 1 (Data Becker, Wiesemann etc.) oder 4 und 12 (Görlitz).

Noch ein kleines Problem kann bei der Druckerinstallation auftauchen. Vielleicht stellen Sie beim Probedrucken fest, daß Ihr Drucker keinen Papiervorschub (Line-Feed) ausgibt.

Je nach Einstellung des Druckers und des Interfaces gibt Ihr Drucker nämlich automatisch nach jedem Wagenrücklauf (Return, CR) ein Line-Feed (LF) aus oder nicht. Es gibt nun zwei Möglichkeiten, den Drucker in diesem Fall zum korrekten Zeilenvorschub zu bewegen:

1. Schalten Sie in Ihrem Drucker den DIP-Schalter ein, der einen automatischen Zeilenvorschub nach jedem CR erzeugt, oder einfacher,
2.) wählen Sie im Installationsmenü die Ausgabe von Line-Feed. Das Programm sendet dann am Ende jeder Zeile LF und CR.

Wenn Sie mit den von uns eingestellten Werten für die Bildgröße einverstanden sind, benötigen Sie keine weiteren Eingaben. Die Frage des Programms nach Änderungen dieser Art können Sie also verneinen. Das Programm legt daraufhin das Maschinenprogramm gebrauchsfertig auf Diskette ab. Anderenfalls werden vom Programm weitere Eingaben zur Bildskalierung der vier möglichen Druckmodi angefordert, bevor es auf Diskette abgelegt wird. Die von uns gewählten Bildgrößen geben das Monitorbild fast unverzerrt auf dem Drucker wieder. Wenn Sie einen exotischen Monitor verwenden, der das Bildverhältnis stark verändert, sollten Sie andere Bildskalierungen austesten.

Die Centronics-Schnittstelle

Noch ein Wort zu Centronics: Diese Schnittstelle besticht durch ihre Sicherheit und Übertragungsgeschwindigkeit. Bei so großen Datenmengen, wie sie für ein HiRes-Bild benötigt werden (etwa 63 bis 250 Blocks auf der Diskette), fällt die Übertragungszeit schon deutlich ins Gewicht. Aber keine Angst, auch über den seriellen Bus reicht die Zeit für eine Hardcopy nicht für eine Kaffeepause! Der Drucker ist so gut wie ausgelastet, aber ein zugeschalteter Puffer kann nicht gefüllt werden. Anders mit Centronics: Hier kann leicht ein Teil des Bildes im Puffer zwischengespeichert werden. Wer ein Centronics-Kabel hat und diesen Vorteil ausnutzen möchte, sollte auf jeden Fall ein vorhandenes Interface ausschalten oder ausbauen. (Für genaue Angaben blättern Sie bitte in der Bedienungsanleitung des Druckers beziehungsweise Interfaces). Die Anschlußbelegung ist identisch mit der Belegung, die im »Komfortablen Treiberprogramm für Centronics-Drucker« (64'er 7/1984, Seite 110) vorgestellt wurde.

User-Port-Pin		Signal
A	(GND)	GND
B	(FLAG)	BUSY
C	(D0)	DATEN
D	(D1)	DATEN
E	(D2)	DATEN
F	(D3)	DATEN
H	(D4)	DATEN
J	(D5)	DATEN
K	(D6)	DATEN
L	(D7)	DATEN
M	(PA2)	STROBE

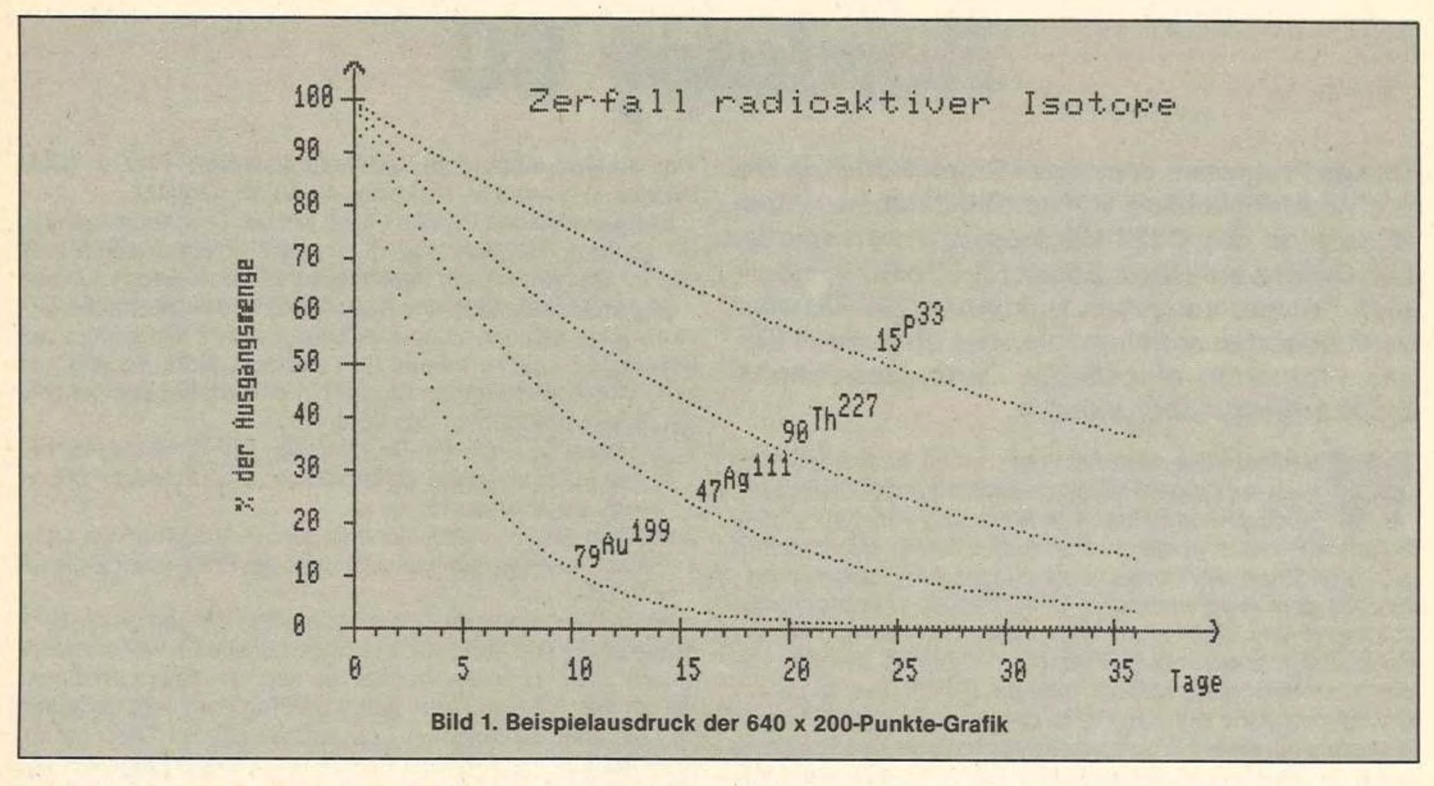


Bild 1. Beispielausdruck der 640 x 200-Punkte-Grafik

Alle Kabel, auch die des seriellen Bus, sollten nur eingesteckt werden, wenn alle Geräte ausgeschaltet sind. Durch diese oft vernachlässigte Regel sind schon viele Bauteile zerstört worden. Zwar hält die von uns eingebaute Schnittstelle den Ein-/Ausgabe-Port auf Eingabe (der Port ist dann hochohmig; damit wird erreicht, daß möglichst nie zwei niederohmige Spannungsquellen aufeinanderstoßen können) und schaltet nur in dem Augenblick der Datenausgabe auf Ausgang, aber gegen einen Kurzschluß oder sonstige gewaltsame »Materialtests«, die bei unsachgemäßer Behandlung auftreten, ist natürlich auch dieser Schutz machtlos.

Aufruf des Programms:

Nachdem das Maschinenprogramm mit

```
BLOAD "HARDCOPY-80.M"
```

geladen wurde, starten Sie die Druckerausgabe mit

```
BANK 0
SYS DEC("F000"),modus
```

Da viele Programmierer jetzt aufstöhnen und »auch auf diesem Computer PEEK, POKE und SYS« hören lassen, möchten wir die Wahl des Aufrufes kurz begründen:

Bei dem Umschreiben von C 64-Programmen auf den C 128 gelang es uns nicht, den Befehl

SYS adresse, argument1, argument2

zum Laufen zu überreden. Erst ein Blick in das ROM brachte die Lösung. Commodore hat auch diesen Befehl überarbeitet und leistungsfähiger gemacht. Der komplette, im Handbuch nicht vollständig dokumentierte Aufruf lautet jetzt:

```
SYS adresse, akku, x-reg, y-reg, prozessorstatus
```

Das heißt, der Interpreter übergibt an das aufgerufene Maschinenprogramm die Prozessor-Register. Alle Argumente können beliebige Ausdrücke sein. Es muß nur gewährleistet werden, daß das jeweilige Ergebnis zwischen 0 und 255, bei der Adresse zwischen 0 und 65535 liegt. Anderenfalls wird ein ILLEGAL QUANTITY ERROR ausgegeben.

Zum anderen wollten wir Ihnen nicht die Möglichkeit nehmen, komplette Grafik-Programme zu compilieren und sie dadurch noch schneller zu machen. Einbinden in Basic wäre zwar auch möglich, aber es ist unwahrscheinlich, daß jeder Compiler diesen Fremdbefehl auch verarbeiten kann. Zudem würde ein Einbinden die Interpreter-Geschwindigkeit herabsetzen, da ein weiterer Befehl berücksichtigt werden muß. Im Gegensatz dazu benutzen wir mit dem SYS-Aufruf einen auf dem C 128 standardmäßig implementierten Befehl, der mit einem guten Compiler ohne weiteres arbeiten sollte.

Aus diesem Grund lautet der Aufruf für das Programm wie oben bereits gezeigt:

```
BANK 0
SYS DEC("F000"),modus
```

Wenn die Angabe in Hex stört, kann natürlich auch mit

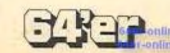
```
BANK 0
SYS 61440,modus
```

die Hardcopy-Routine starten.

»modus« ist irgend ein arithmetischer Ausdruck, der einen Wert zwischen 1 und 4 liefert. Bei Werten größer als 255 weigert sich der Interpreter die Zahl in eine 8-Bit-Zahl umzurechnen, bei allen sonstigen nicht erlaubten Zahlen meldet sich das Programm mit ILLEGAL QUANTITY und zwei Piepstönen zurück. Die Ausgabe der Fehlermeldung läßt sich durch die TRAP-Anweisung abfangen. So ist es möglich, auf Fehler zu reagieren, um zum Beispiel vom HiRes- in den Textmodus zu schalten. Wird das Programm ordnungsgemäß verlassen, ertönt ein Piepston. Auf die Ausgabe einer optischen Meldung wurde verzichtet, um damit den HiRes-Bildschirm nicht zu zerstören.

Noch eine Anmerkung zum Schluß: In ausführlichen Tests des Programms mit verschiedenen Druckern und Interfaces hat sich herausgestellt, daß das Görlitz-Interface VC 2.6 für den Epson FX-80 nicht ganz korrekt mit dem Bus des C 128 zusammenarbeitet. Mit diesem Interface konnte zwar die unter »1« voreingestellte Bildgröße fehlerfrei ausgegeben werden, wählt man aber die unter »3« voreingestellte Bildgröße, dann geht die Steuercode-Sequenz »ESC * $03 $80 $07« auf dem Bus verloren. Das Interface scheint hier als Aktivfilter zu wirken. Problemlos lief Hardcopy 80 zum Beispiel mit dem Data Becker-Interface VCI 2.8 (Sekundäradresse 1). Auch die Centronics-Schnittstelle hat auf keinem der bisher getesteten Drucker versagt. Sollte Hardcopy 80 einmal nicht auf Anhieb funktionieren, dann werfen Sie die Flinte nicht gleich ins Korn, probieren Sie verschiedene Druckdichten und Sekundäradressen aus; die Mühe lohnt sich bestimmt (siehe Bild) und das Ergebnis wird Hindernisse auf dem Weg zum Erfolg rasch vergessen lassen.

(Dieter Winkler/Thomas Rumbach/og)



```
1 REM "{25SPACE}HARDCOPY-80
2 :
3 REM "{4SPACE}MASCHINENROUTINE ZUR AUSGABE
  DES 80-ZEICHEN HIRES-BILDSCHIRMS DES
4 REM "{11SPACE}C-128 AUF EINEN FX-80 ODER K
  OMPATIBLEN DRUCKER
5 :
6 REM "{28SPACE}BY{2SPACE}KRW
7 :
8 REM "{28SPACE}10/1985
9 :
10 :
11 :
100 FAST : BANK 0
110 QD$=CHR$(17)   : REM "CURSOR DOWN
120 CL$=CHR$(147) : REM "SCREEN CLEAR
130 :
140 PRINT CL$; TAB(30); "HARDCOPY-80"
150 PRINT TAB(30); QD$; "(C){2SPACE}BY KRW"
160 PRINT TAB(20); QD$; QD$; "* DATAZEILEN L
    ESEN"
170 FOR SP=DEC("F000") TO DEC("F33D")
180 READ CO: POKE SP,CO
190 NEXT SP
200 :
210 PRINT TAB(20); QD$; "* INSTALLATION"
220 PRINT TAB(25); QD$; "DIE DRUCKERADRESSE
    IST"; PEEK(DEC("F10A"))
230 PRINT TAB(25); "DIE SEKUNDFRADRESSE IST"
    ; PEEK(DEC("F0F7"))
240 PRINT TAB(25); "LINEFEED WIRD ZUM DRUCKE
    R GESANDT: ";
250 IF PEEK(DEC("F220"))=5 THEN PRINT "NEIN"
    : ELSE PRINT "JA"
260 PRINT TAB(20); QD$; QD$; "+NDERUNGEN GEW
    XNSCHT ?{2SPACE}(J/{RVSON}N{RVOFF}) ";
270 GET KEY WA$
280 IF WA$="N" OR WA$=CHR$(13) THEN PRINT "N
    EIN": GOTO 640
290 IF WA$<>"J" THEN 270
300 PRINT "JA"
310 PRINT TAB(20); QD$; "EINGABE DER DRUCKER
    ADRESSE (ODER 'C' FUR CENTRONICS) ";
320 INPUT AD$
330 IF AD$<>"C" AND (VAL(AD$)<4 OR VAL(AD$)>
    15) THEN 320
340 IF AD$="C" THEN BEGIN
350 : POKE DEC("F0F7"),255
360 BEND : ELSE BEGIN
370 : PRINT TAB(20); "EINGABE DER SEKUNDFRAD
    RESSE ";
380 : INPUT SA$
390 : IF VAL(SA$)<0 OR VAL(SA$)>15 THEN 380
400 : POKE DEC("F10A"),VAL(AD$)
410 : POKE DEC("F0F7"),VAL(SA$)
420 BEND
430 PRINT TAB(20); "SOLL LINEFEED GESENDET W
    ERDEN (J/{RVSON}N{RVOFF}) ";
440 GET KEY WA$
450 IF WA$="J" THEN BEGIN
460 PRINT "JA"
470 POKE DEC("F220"),6
480 BEND : ELSE BEGIN
490 IF WA$<>"N" AND WA$<>CHR$(13) THEN 440
500 PRINT "NEIN"
510 POKE DEC("F220"),5
520 BEND
530 PRINT TAB(20); QD$; "ANGABEN RICHTIG ?{2
    SPACE}({RVSON}J{RVOFF}/N) ";
540 GET KEY WA$
550 IF WA$=CHR$(13) OR WA$="J" THEN PRINT "J
    A": GOTO 640
560 IF WA$="N" THEN BEGIN
570 : PRINT "NEIN"
580 : GOTO 220
590 BEND : ELSE BEGIN
600 : GOTO 540
610 BEND
620 :
630 :
640 PRINT CL$; TAB(20); "* +NDERUNG DER BILD
    AUSGABEPARAMETER (J/{RVSON}N{RVOFF}) ";
650 GET KEY WA$
660 IF WA$="N" OR WA$=CHR$(13) THEN PRINT "N
    EIN": GOTO 1190
670 IF WA$<>"J" THEN 650
680 PRINT "JA"
690 PRINT TAB(25); QD$; "DRUCKMODUS ('0' ->
    ENDE DER EINGABEN) ";
700 GET KEY NR$
710 BN=VAL(NR$)
720 IF ((BN=0 AND NR$<>"0") OR BN>4) THEN 70
    0
730 PRINT NR$
740 IF BN=0 THEN 1190
750 CL=PEEK(DEC("F323")+BN-1)
760 XF=CL AND 3
770 YF=CL AND 192
780 RV=CL AND 32
790 IF (YF AND 128)=0 THEN BEGIN
800 : YF=1
810 BEND : ELSE BEGIN
820 : IF (YF AND 64)=0 THEN BEGIN
830 : : YF=2
840 : BEND : ELSE BEGIN
850 : : YF=4
860 : BEND
870 BEND
880 FX=PEEK(DEC("F31B")+BN-1)
890 PRINT TAB(25); QD$; QD$; "DRUCKMODUS: ";
    BN
900 PRINT TAB(25); QD$; "VERVIELFACHEN IN X-
    RICHTUNG: "; XF
910 PRINT TAB(25); "VERVIELFACHEN IN Y-RICHT
    UNG: "; YF
920 PRINT TAB(25); "REVERSE ON:{18SPACE}";
930 IF RV THEN PRINT "JA": ELSE PRINT "NEIN"
940 PRINT TAB(25); "PUNKTDICHTE{4SPACE}('ESC
    * N') : "; FX
950 PRINT TAB(20); QD$; QD$; "NEUE VERVIELFA
    CHUNG IN X-RICHTUNG: (1,2,3) ";
960 GET KEY WA$
970 IF WA$<"1" OR WA$>"3" THEN 960
980 PRINT WA$
990 CL=VAL(WA$)
1000 PRINT TAB(20); "NEUE VERVIELFACHUNG IN
     Y-RICHTUNG: (1,2,4) ";
1010 GET KEY WA$
1020 IF WA$<>"1" AND WA$<>"2" AND WA$<>"4" T
     HEN 1010
1030 PRINT WA$
1040 IF WA$="4" THEN CL=CL OR 192
1050 IF WA$="2" THEN CL=CL OR 128
1060 PRINT TAB(20); "REVERS (J/N):{30SPACE}"
     ;
1070 GET KEY WA$
1080 IF WA$<>"J" AND WA$<>"N" THEN 1070
1090 IF WA$="J" THEN CL=CL OR 32: PRINT "JA"
     : ELSE PRINT "NEIN"
1100 POKE DEC("F323")+BN-1,CL
1110 PRINT TAB(20); "NEUE PUNKTDICHTE FXR FX
     -80:{2SPACE}(0-7){9SPACE}";
1120 GET KEY WA$
1130 IF WA$<"0" OR WA$>"7" THEN 1120
1140 FX=VAL(WA$)
1150 POKE DEC("F31B")+BN-1,FX
1160 PRINT FX
1170 GOTO 690
1180 :
1190 PRINT TAB(20); QD$; QD$; "* ABSPEICHERN
      DER MASCHINENROUTINE"
1200 PRINT TAB(20); QD$; "BITTE DISKETTE IN
     LAUFWERK 8 EINLEGEN (TASTE)"
1210 GET KEY WA$
```

Listing 1. »Hardcopy 80«

```
1220 SCRATCH "HARDCOPY-80.M"
1230 IF DS>1 THEN PRINT DS$: STOP
1240 BSAVE "HARDCOPY-80.M",U8,ON B0,P(DEC("F
     000")) TO P(DEC("F33E"))
1250 IF DS THEN PRINT DS$
1260 :
2000 :
2010 DATA 141,67,243,32,204,242,162,22,189,3
     9,243,157,0,12,202,16,247,174,67
2020 :
2030 DATA 243,240,4,224,5,144,87,162,204,142
     ,13,12,162,255,142,14,12,32,0,12
2040 :
2050 DATA 169,7,162,210,142,13,12,162,255,14
     2,14,12,32,0,12,162,0,160,0,136
2060 :
2070 DATA 208,253,202,208,250,56,8,169,7,162
     ,210,142,13,12,162,255,142,14,12
2080 :
2090 DATA  32,0,12,169,4,162,195,142,13,12,1
     62,255,142,14,12,32,0,12,40,176
2100 :
2110 DATA   1,96,162,40,160,125,142,13,12,14
     0,14,12,76,0,12,202,189,31,243,72
2120 :
2130 DATA 189,27,243,141,53,242,189,35,243,1
     41,63,243,41,3,170,160,128,169
2140 :
2150 DATA   2,140,52,242,141,51,242,202,240,
     19,24,152,109,52,242,141,52,242,173
2160 :
2170 DATA  51,242,105,2,141,51,242,202,208,2
     37,160,160,169,0,140,73,241,141
2180 :
2190 DATA  81,241,162,100,142,66,243,44,63,2
     43,8,16,2,112,9,78,66,243,14,73
2200 :
2210 DATA 241,46,81,241,40,48,9,78,66,243,14
     ,73,241,46,81,241,172,73,241,173
2220 :
2230 DATA  81,241,140,96,241,141,98,241,169,
     0,141,64,243,141,65,243,169,0,162
2240 :
2250 DATA 189,142,13,12,162,255,142,14,12,32
     ,0,12,169,4,160,12,140,69,243,192
2260 :
2270 DATA 255,240,49,162,186,142,13,12,162,2
     55,142,14,12,162,5,32,0,12,162
2280 :
2290 DATA 192,142,13,12,162,255,142,14,12,32
     ,0,12,144,4,104,76,26,240,162,4
2300 :
2310 DATA 169,201,141,13,12,169,255,141,14,1
     2,32,0,12,169,27,32,123,242,169
2320 :
2330 DATA  51,32,123,242,104,32,123,242,32,3
     1,242,76,85,241,24,173,64,243,105
2340 :
2350 DATA   0,141,64,243,173,65,243,105,0,14
     1,65,243,169,72,162,243,141,147,241
2360 :
2370 DATA 142,148,241,169,0,160,0,141,70,243
     ,140,71,243,162,18,173,65,243,160
2380 :
2390 DATA 204,140,13,12,160,205,140,14,12,32
     ,0,12,162,19,173,64,243,32,0,12
2400 :
2410 DATA 162,31,160,218,140,13,12,160,205,1
     40,14,12,32,0,12,141,255,255,238
2420 :
2430 DATA 147,241,208,3,238,148,241,56,173,7
     0,243,233,1,141,70,243,173,71,243
2440 :
2450 DATA 233,0,141,71,243,176,223,162,0,160
     ,8,169,0,141,62,243,62,72,243,46
2460 :
2470 DATA  62,243,62,152,243,46,62,243,44,63
     ,243,16,2,112,41,62,232,243,46,62
2480 :
2490 DATA 243,62,56,244,46,62,243,44,63,243,
     48,24,62,136,244,46,62,243,62,216
2500 :
2510 DATA 244,46,62,243,62,40,245,46,62,243,
     62,120,245,46,62,243,173,62,243
2520 :
2530 DATA  32,58,242,136,208,182,232,224,80,
     208,175,206,66,243,240,6,32,31,242
2540 :
2550 DATA  76,68,241,169,204,141,13,12,169,2
     55,141,14,12,32,0,12,24,76,65,240
2560 :
2570 DATA 160,5,140,71,243,172,71,243,185,51
     ,242,32,123,242,206,71,243,16,242
2580 :
2590 DATA  96,0,0,0,42,27,13,10,142,67,243,1
     40,68,243,44,63,243,16,11,170,112
2600 :
2610 DATA   5,189,7,243,80,3,189,23,243,141,
     62,243,173,63,243,72,41,3,141,70
2620 :
2630 DATA 243,104,41,32,240,8,173,62,243,73,
     255,141,62,243,173,62,243,32,123
2640 :
2650 DATA 242,206,70,243,208,245,174,67,243,
     172,68,243,96,72,173,69,243,201
2660 :
2670 DATA 255,240,14,169,210,141,13,12,169,2
     55,141,14,12,104,76,0,12,162,3
2680 :
2690 DATA 160,221,32,247,242,104,162,1,160,2
     21,32,247,242,162,13,160,221,32
2700 :
2710 DATA 231,242,162,0,160,221,32,231,242,4
     1,251,32,251,242,9,4,32,251,242
2720 :
2730 DATA 162,13,160,221,32,231,242,41,16,24
     0,245,169,0,162,3,160,221,76,247
2740 :
2750 DATA 242,32,195,242,162,2,160,221,32,23
     1,242,9,4,32,251,242,162,0,160
2760 :
2770 DATA 221,32,231,242,9,4,76,251,242,134,
     252,132,253,160,0,162,252,142,170
2780 :
2790 DATA   2,162,0,76,162,2,134,252,132,253
     ,160,0,162,252,142,185,2,162,0,76
2800 :
2810 DATA 175,2,0,3,12,15,48,51,60,63,192,19
     5,204,207,240,243,252,255,0,15
2820 :
2830 DATA 240,255,7,7,3,3,23,23,23,23,1,33,1
     31,163,72,173,0,255,133,254,169,0
2840 :
2850 DATA 141,0,255,104,32,255,255,72,165,25
     4,141,0,255,104,96
2860 :
2870 END
2880 :
60000 ZZ$="HARDCOPY-80": UN=8
60010 OPEN 15,UN,15,"S0:"+ZZ$
60020 GOSUB 60100
60030 SAVE ZZ$,UN
60040 GOSUB 60100
60050 VERIFY ZZ$,UN
60060 CLOSE 15
60070 END
60080 :
60100 INPUT#15, S1, S$, S2, S3
60110 IF S1=1 THEN PRINT S2; S$
60120 IF S1<20 THEN RETURN
60130 PRINT S1 ", " S$ "," S2; "," S3
60140 CLOSE 15
```

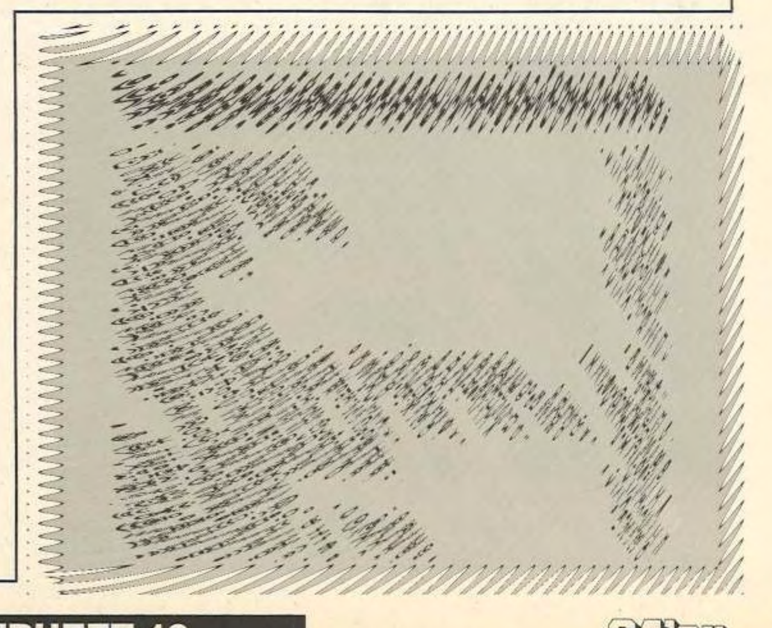
Listing 1. »Hardcopy 80«-Programm zum Ausdrucken der 640 x 200-Punkte-Grafik auf dem C 128

```
1 rem "                         Hardcopy-80
2 :
3 rem "             Routine erstellt Hires-Bilder auf FX-80
4 :
5 rem "                          by KRW
6 :
7 rem "                          1985
8 :
9 rem "                        Version 1.4
10 :
11 :
50 zz$="hardcopy-80.m"
60 open 15,8,15,"s0:"+zz$
70 gosub 60100
80 open 2,8,2,zz$+",p,w"
90 gosub 60100
100 :
110 sys 9*4096
120 .opt o2
130 ;
140 ; "*** Konstante ***"
150 ;
160 nrcmd   = 4              ; "Anzahl der Befehle
170 bell    = 7              ; "Klingelzeichen
180 lf      = 10             ; "Linefeed
190 cr      = 13             ; "Return
200 esc     = 27             ; "Escape
210 fa      = 4              ; "Geräteadresse des Druckers
220 sa      = 12             ; "Sekundäradresse des Druckers
230 ;
240 ; "*** Speicherstellen ***"
250 ;
260 savadr  = $fc            ; "Adressübergabe (2 Byte)
270 savcnfg = $fe            ; "MMU-Registerzwischenspeicher
280 ldpnt   = $02aa
290 stpnt   = $02b9
300 ;
310 ; "*** Kernalroutinen ***"
320 ;
330 setnam  = $ffbd
340 setlfs  = $ffba
350 open    = $ffc0
360 close   = $ffc3
370 clrchn  = $ffcc
380 chkout  = $ffc9
390 chrout  = $ffd2
400 ;
410 quant   = $7d28
420 ;
430 ldfar   = $02a2          ; "von anderer Bank laden
440 stfar   = $02af          ; "in andere Bank speichern
450 ;
980 *= $f000
990 ;
1000 init     sta savx      ; "gewünschten Modus zwischenspeichern
1010 ^        jsr incent    ; "Initialisierung der Centronicsschnittstelle
1020 ^        ldx #endfar-jsfar
1030 copyfar  lda jsfar,x   ; "Sprungroutine in gemeinsamen
1040 ^        sta $0c00,x   ; "Speicherbereich kopieren
1050 ^        dex
1060 ^        bpl copyfar
1070 ^        ldx savx      ; "Modus zurückholen
1080 ^        beq error     ; "=0 -> Fehler
1090 ^        cpx #nrcmd+1  ; "Anzahl der Druckmodi+1
1100 ^        bcc init1     ; "kleiner -> erlaubt
1110 error    ldx #<clrchn  ; "I/O auf Standard
1120 ^        stx $0c0d
1130 ^        ldx #>clrchn
1140 ^        stx $0c0e
1150 ^        jsr $0c00
1160 ^        lda #bell     ; "2* Bell -> Fehler
1170 ^        ldx #<chrout  ; "1* Bell -> Ende OK
1180 ^        stx $0c0d
1190 ^        ldx #>chrout
1200 ^        stx $0c0e
1210 ^        jsr $0c00
1220 ^        ldx #0        ; "Warteschleife für Piepton
1230 ^        ldy #0
1240 err1     dey
1250 ^        bne err1
1260 ^        dex
1270 ^        bne err1
1280 ^        sec           ; "Fehler
1290 ende     php           ; "Einsprung zum Beenden der Routine
1300 ^        lda #bell
1310 ^        ldx #<chrout
1320 ^        stx $0c0d
1330 ^        ldx #>chrout
1340 ^        stx $0c0e
1350 ^        jsr $0c00
1360 ^        lda #4        ; "Druckerfile schließen
1370 ^        ldx #<close   ; "und zurück ins BASIC
1380 ^        stx $0c0d
1390 ^        ldx #>close
1400 ^        stx $0c0e
1410 ^        jsr $0c00
1420 ^        plp           ; "Fehlerstatus vom Stack holen
1430 ^        bcs ende1     ; "Fehlerabbruch
1440 ^        rts           ; "fehlerfrei
1450 ;
1460 ende1    ldx #<quant   ; "Illegal Quantity
1470 ^        ldy #>quant
1480 ^        stx $0c0d
1490 ^        sty $0c0e
1500 ^        jmp $0c00
1510 ;
1520 init1    dex           ; "x als Zeiger in Tabellen
1530 ^        lda prtvor,x  ; "Papiervorschub des Druckers
1540 ^        pha           ; "retten
1550 ^        lda prtctl,x  ; "Druckersteuerung
1560 ^        sta prtcmd1
1570 ^        lda ctrl,x    ; "Programmsteuerbyte
1580 ^        sta steuer
1590 ^        and #%00000011 ; "aus x-Vervielfachung Anzahl
1600 ^        tax           ; "der zu Uebertragenden Bytes pro Zeile errechne
1610 ^        ldy #<640     ; "Wert fuer keine Vervielfachung
1620 ^        lda #>640
1630 ^        sty prtcmd+1
1640 ^        sta prtcmd
1650 ^        dex
1660 ^        beq init3     ; "keine Vervielfachung
1670 init2    clc           ; "mal Vervielfachung
1680 ^        tya           ; "Low-Byte
1690 ^        adc prtcmd+1
1700 ^        sta prtcmd+1
1710 ^        lda prtcmd
1720 ^        adc #>640
1730 ^        sta prtcmd
1740 ^        dex
1750 ^        bne init2
1760 init3    ldy #<160     ; "aus y-Vervielfachung Anzahl
1770 ^        lda #>160     ; "der zu kopierenden Bytes errechnen
1780 ^        sty copyl+1
1790 ^        sta copyh+1
1800 ^        ldx #100      ; "max. Anzahl der Druckerzeilen
1810 ^        stx anzzei    ; "in Zaehler
1820 ^        bit steuer
1830 ^        php
1840 ^        bpl init31    ; "keine Vervielfachung
1850 ^        bvs init4     ; "v=1 -> y=y*2
1860 init31   lsr anzzei    ; "Zeilen /2
1870 ^        asl copyl+1   ; "Anzahl *2
1880 ^        rol copyh+1
1890 init4    plp
1900 ^        bmi init5     ; "n=1 -> y=y*2*2
1910 ^        lsr anzzei
1920 ^        asl copyl+1
1930 ^        rol copyh+1
1940 init5    ldy copyl+1   ; "errechnete Anzahl in Zähler uebernehmen
1950 ^        lda copyh+1
1960 ^        sty countl+1
1970 ^        sta counth+1
1980 ^        lda #0
1990 ^        sta scrpnt    ; "Bildschirmzeiger auf Anfang Bildschirm
2000 ^        sta scrpnt+1
2010 ^        lda #0        ; "Druckerkanal öffnen
2020 ^        ldx #<setnam  ; "Länge des Namen = 0
2030 ^        stx $0c0d
2040 ^        ldx #>setnam
2050 ^        stx $0c0e
2060 ^        jsr $0c00
2070 ^        lda #4        ; "logische Filenummer (la)
2080 ^        ldy #sa       ; "Sekundäradresse (sa)
2090 ^        sty sekadr
2100 ^        cpy #$ff      ; "Centronics ?
2110 ^        beq in2       ; "ja -> kein File auf seriellem Bus öffnen
2120 ^        ldx #<setlfs
2130 ^        stx $0c0d
2140 ^        ldx #>setlfs
2150 ^        stx $0c0e
2160 ^        ldx #fa       ; "Geräteadresse (fa)
2170 ^        jsr $0c00
2180 ^        ldx #<open    ; "File öffnen
2190 ^        stx $0c0d
2200 ^        ldx #>open
2210 ^        stx $0c0e
2220 ^        jsr $0c00
2230 ^        bcc in12      ; "Fehler bei OPEN aufgetreten -> Abbruch
2240 ^        pla           ; "Papiervorschubsteuerung vom Stack
2250 ^        jmp error
2260 ;
2270 in12     ldx #4        ; "Ausgabe auf Drucker umlenken
2280 ^        lda #<chkout
2290 ^        sta $0c0d
2300 ^        lda #>chkout
2310 ^        sta $0c0e
2320 ^        jsr $0c00
2330 in2      lda #esc      ; "Druckervorschub einstellen
2340 ^        jsr aufbus
2350 ^        lda #"3"      ; "Vorschub = nn/216 Zoll
2360 ^        jsr aufbus
2370 ^        pla
2380 ^        jsr aufbus    ; "nn ausgeben
2390 ^        jsr drucker   ; "Drucker in Bitmode schalten
2400 ^        jmp copy1
2410 ;
2420 copy     clc           ; "Zeiger für Bildschirmram erhöhen
2430          lda scrpnt
2440 copyl    adc #00       ; "wird bei init eingesetzt
2450 ^        sta scrpnt
2460 ^        lda scrpnt+1
2470 copyh    adc #00
2480 ^        sta scrpnt+1
2490 copy1    lda #<buffer  ; "Zeiger auf Anfang des Buffers
2500 ^        ldx #>buffer
2510 ^        sta bufpnt+1
2520 ^        stx bufpnt+2
2530 countl   lda #$00      ; "Anzahl der Bytes (low Byte)
2540 counth   ldy #$00      ; "Anzahl der Bytes (High Byte)
2550 ^        sta counter
2560 ^        sty counter+1
2570 ;
2580 ; "*** Bytes aus Bildschirmspeicher ins RAM kopieren ***"
2590 ;
2600 ^        ldx #18
2610 ^        lda scrpnt+1  ; "Bildschirm-RAM-Adresse (High Byte)
2620 ^        ldy #<$cdcc
2630 ^        sty $0c0d
2640 ^        ldy #>$cdcc
2650 ^        sty $0c0e
2660 ^        jsr $0c00
2670 ^        ldx #19
2680 ^        lda scrpnt    ; "Bildschirm-RAM-Adresse (Low Byte)
2690 ^        jsr $0c00
2700 ^        ldx #31
2710 ^        ldy #<$cdda
2720 ^        sty $0c0d
2730 ^        ldy #>$cdda
2740 ^        sty $0c0e
2750 copy2    jsr $0c00     ; "Byte holen
2760 bufpnt   sta $ffff     ; "und ins RAM speichern (Adresse wird eingesetzt)
2770 ^        inc bufpnt+1
2780 ^        bne copy3
2790 ^        inc bufpnt+2
2800 copy3    sec           ; "Zähler herrunterzählen
2810 ^        lda counter
2820 ^        sbc #1
2830 ^        sta counter
2840 ^        lda counter+1
2850 ^        sbc #0
2860 ^        sta counter+1
2870 ^        bcs copy2
2880 ;
2890 ; "*** Ins RAM kopierte Bytes verarbeiten ***"
2900 ;
2910 ^        ldx #0        ; "Byte in der aktuellen Zeile
2920 pr1      ldy #8        ; "8 Bit pro Byte
2930 pr2      lda #0        ; "Data löschen
2940 ^        sta data
2950 ^        rol row1,x
2960 ^        rol data
2970 ^        rol row2,x
2980 ^        rol data
2990 ^        bit steuer    ; "vervielfachen ?
3000 ^        bpl pr3       ; "keine Vergrößerung in y-Richtung
3010 ^        bvs out       ; "*4
3020 pr3      rol row3,x
3030 ^        rol data
3040 ^        rol row4,x
3050 ^        rol data
3060 ^        bit steuer
3070 ^        bmi out       ; "*2
3080 ^        rol row5,x
3090 ^        rol data
```

Listing 2. Der kommentierte Quelltext der 640 x 200-Punkte-Grafik-Hardcopy für den C128

```
3100 ^        rol row6,x
3110 ^        rol data
3120 ^        rol row7,x
3130 ^        rol data
3140 ^        rol row8,x
3150 ^        rol data
3160 out      lda data       ; "fertiges Byte laden
3170 ^        jsr ausgabe    ; "Bits verarbeiten und ausgeben
3180 ^        dey            ; "nächstes Bit
3190 ^        bne pr2        ; "Byte noch nicht fertig
3200 ^        inx            ; "nächstes Byte in dieser Zeile
3210 ^        cpx #80        ; "alle 80 Byte dieser Zeile fertig ?
3220 ^        bne pr1        ; "nein
3230 ^        dec anzzei     ; "ja -> Zeilenzähler erniedrigen
3240 ^        beq spa1       ; "= 0 -> fertig
3250 ^        jsr drucker    ; "sonst Drucker für neue Zeile anweisen
3260 ^        jmp copy       ; "neue Zeile bearbeiten
3270 ;
3280 spa1     lda #<clrchn
3290 ^        sta $0c0d
3300 ^        lda #>clrchn
3310 ^        sta $0c0e
3320 ^        jsr $0c00
3330 ^        clc            ; "Ende fehlerfrei
3340 ^        jmp ende
3350 ;
3360 ; "*** Druckeransteuerung ***"
3370 ;
3380 drucker  ldy #5         ; "Anzahl der Befehlbytes
3390 ^        sty counter+1
3400 dr1      ldy counter+1
3410 ^        lda prtcmd,y
3420 ^        jsr aufbus     ; "an Drucker senden
3430 ^        dec counter+1
3440 ^        bpl dr1
3450 ^        rts
3460 ;
3470 prtcmd   .byt 0, 0      ; "Anzahl Punkte pro Zeile (wird eingesetzt)
3480 prtcmd1  .byt 0, "*", esc, cr, lf; "Druckmodus (wird eingesetzt)
3490 ;
3500 ; "*** Ausgabensteuerung ***"
3510 ;
3520 ausgabe  stx savx       ; "x retten
3530 ^        sty savy       ; "y retten
3540 ^        bit steuer
3550 ^        bpl aus2       ; "kein Vervielfachen in y-Richtung
3560 ^        tax
3570 ^        bvs aus1
3580 ^        lda doppel,x   ; "y=y*2
3590 ^        bvc aus2       ; "unbedingter Sprung
3600 ;
3610 aus1     lda vier,x     ; "y=y*4
3620 aus2     sta data       ; "Byte zwischenspeichern
3630 ^        lda steuer
3640 ^        pha
3650 ^        and #%00000011 ; "Vervielfachung in x-Richtung
3660 ^        sta counter    ; "als Zähler
3670 ^        pla
3680 ^        and #%00100000 ; "REVERSE
3690 ^        beq aus3       ; "nein
3700 ^        lda data
3710 ^        eor #$ff
3720 ^        sta data
3730 aus3     lda data
3740 ^        jsr aufbus     ; "an Drucker
3750 ^        dec counter
3760 ^        bne aus3
3770 ^        ldx savx       ; "Register zurückspeichern.
3780 ^        ldy savy
3790 ^        rts
3800 ;
3810 ; "*** Byte ausgeben (seriell/centronics) ***"
3820 ;
3830 aufbus   pha
3840 ^        lda sekadr     ; "Centronics ?
3850 ^        cmp #$ff
3860 ^        beq auf1       ; "ja
3870 ^        lda #<chrout
3880 ^        sta $0c0d
3890 ^        lda #>chrout
3900 ^        sta $0c0e
3910 ^        pla            ; "Byte zurückholen
3920 ^        jmp $0c00      ; "und ausgeben (seriell)
3930 ;
3940 auf1     ldx #<$dd03    ; "DDRB auf Ausgabe
3950 ^        ldy #>$dd03
3960 ^        jsr indst
3970 ^        pla            ; "Byte zurückholen
3980 ^        ldx #<$dd01    ; "PRB
3990 ^        ldy #>$dd01
4000 ^        jsr indst
4010 ^        ldx #<$dd0d    ; "IRQ löschen
4020 ^        ldy #>$dd0d
4030 ^        jsr indld
4040 ^        ldx #<$dd00    ; "Strobe
4050 ^        ldy #>$dd00
4060 ^        jsr indld
4070 ^        and #$fb
4080 ^        jsr indst1
4090 ^        ora #$04
4100 ^        jsr indst1
4110 auf2     ldx #<$dd0d    ; "auf Druckerantwort warten
4120 ^        ldy #>$dd0d
4130 ^        jsr indld
4140 ^        and #$10
4150 ^        beq auf2
4160 initcia  lda #0         ; "DDRB (sicherheitshalber) auf Eingang
4170 ^        ldx #<$dd03
4180 ^        ldy #>$dd03
4190 ^        jmp indst
4200 ;
4210 ; "*** Init Centronics ***"
4220 ;
4230 incent   jsr initcia    ; "Centronics initialisieren
4240 ^        ldx #<$dd02
4250 ^        ldy #>$dd02
4260 ^        jsr indld      ; "Strobebit
4270 ^        ora #%00000100 ; "auf Ausgabe
4280 ^        jsr indst1
4290 ^        ldx #<$dd00
4300 ^        ldy #>$dd00
4310 ^        jsr indld      ; "Strobepegel
4320 ^        ora #%00000100 ; "auf high
4330 ^        jmp indst1
4340 ;
4350 ;
4360 ; "*** aus anderer Bank laden ***"
4370 ;
4380 indld    stx savadr
4390 ^        sty savadr+1
4400 indld1   ldy #$00    "/7
4410 ^        ldx #savadr
4420 ^        stx ldpnt
4430 ^        ldx #$00       ; "Bank 15 & I/O einblenden
4440 ^        jmp ldfar
4450 ;
4460 ; "*** in andere Bank speichern ***"
4470 ;
4480 indst    stx savadr
4490 ^        sty savadr+1
4500 indst1   ldy #$00
4510 ^        ldx #savadr
4520 ^        stx stpnt
4530 ^        ldx #$00       ; "Bank 15 & I/O einblenden
4540 ^        jmp stfar
4550 ;
4560 ; "*** Tabellen ***"
4570 ;
4580 doppel   .byt %00000000, %00000011, %00001100, %00001111
4590 ^        .byt %00110000, %00110011, %00111100, %00111111
4600 ^        .byt %11000000, %11000011, %11001100, %11001111
4610 ^        .byt %11110000, %11110011, %11111100, %11111111
4620 ;
4630 vier     .byt %00000000, %00001111, %11110000, %11111111
4640 ;
4650 prtctl   .byt 7, 7
4660 ^        .byt 3, 3
4670 ;
4680 prtvor   .byt 23, 23
4690 ^        .byt 23, 23
4700 ;
4710 ctrl     .byt %00000001, %00100001
4720 ^        .byt %10000011, %10100011
4730 ;
4740 ; "Erklärung der Steuerbits (ctrl-Tabelle)
4750 ;
4760 ;" bit 7  = 0 : normal
4770 ;"        = 1 : vervielfachen in y-Richtung
4780 ;" bit 6  = 0 : y=y*2 (nur gültig  wenn bit 7 = 1)
4790 ;"        = 1 : y=y*4 (nur gültig  wenn bit 7 = 1)"
4800 ;" bit 5  = 0 : normal
4810 ;"        = 1 : reverse
4820 ;" bit 0/1     : vervielfachen in x (0 bis 3 mal; 0 = kein Ausdruck)
4830 ;
4840 ;  "*** jsr far (wird nach $0c00 verschoben) ***"
4850 ;
4860 jsfar    pha
4870 ^        lda $ff00      ; "aktuelle Konfiguration retten
4880 ^        sta savcnfg
4890 ^        lda #$00       ; "Bank 15 & I/O einblenden
4900 ^        sta $ff00
4910 ^        pla
4920 ^        jsr $ffff      ; "wird überschrieben
4930 ^        pha
4940 ^        lda savcnfg    ; "alte Konfiguration wiederherstellen
4950 ^        sta $ff00
4960 ^        pla
4970 endfar   rts
4980 ;
4990 data     *= *+1
5000 steuer   *= *+1          ; "Kontrollwort
5010 scrpnt   *= *+2          ; "Bildschirmzeiger
5020 anzzei   *= *+1          ; "Anzahl der Bildschirmzeilen
5030 savx     *= *+1          ; "Zwischenspeicher fuer x-Register
5040 savy     *= *+1          ; "Zwischenspeicher fuer y-Register
5050 sekadr   *= *+1          ; "Sekundäradresse
5060 counter  *= *+2          ; "Zähler fuer verschiedene Zwecke
5070 buffer   =*
5080 row1     = buffer + (0*80)
5090 row2     = buffer + (1*80)
5100 row3     = buffer + (2*80)
5110 row4     = buffer + (3*80)
5120 row5     = buffer + (4*80)
5130 row6     = buffer + (5*80)
5140 row7     = buffer + (6*80)
5150 row8     = buffer + (7*80)
5160 ;
59980 .end
59990 end
60000 zz$="hardcopy-80.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60080 :
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15
```

Listing 2. Der kommentierte Quelltext von »Hardcopy 80« (Schluß)



Kochrezepte: Kernel-Routinen für jeden Zweck

Um komfortabel und schnell Programme in Assembler zu schreiben, müssen Sie sich im Betriebssystem auskennen. Wir zeigen Ihnen, wo die wichtigen Routinen liegen und wie man sie anwendet.

Wenn Sie häufig Programme in Assembler schreiben, ist die im C128 integrierte Software wie Betriebssystem und Basic-Interpreter für die eigene Programmentwicklung sehr hilfreich. So gut wie alle Routinen lassen sich von eigenen Maschinenprogrammen aus nutzen. Aber wie? Wie werden die Routinen aufgerufen, welche Parameter müssen ihnen übergeben werden, wie werden sie angewendet...? Die offenen Fragen, die auf den Programmierer zukommen, wachsen schnell ins Uferlose. Damit Sie nicht enttäuscht den C128 in die Ecke stellen oder nur noch in Basic programmieren, geben wir Ihnen wertvolle Tips und Tricks, die Ihnen beim Programmieren in Assembler helfen.

Der Bildschirm-Editor

Um Programme oder Daten komfortabel eingeben zu können, enthält das Betriebssystem des C128 einen Bildschirm-Editor, der – im Gegensatz zu vielen anderen Mikrocomputern – ein sogenannter »Fullscreen-Editor« ist. Dieser Typ eines Bildschirm-Editors erlaubt es dem Anwender, sich mit dem Cursor völlig frei auf dem Bildschirm zu bewegen, hier und dort Änderungen am Programm vorzunehmen, die beim Drücken der <RETURN>- beziehungsweise <ENTER>-Taste sofort ausgeführt werden.

Das Gegenstück hierzu, der sogenannte »Line-Editor«, erlaubt Cursor-Bewegungen und damit auch Änderungen nur innerhalb der aktuellen Bildschirmzeile. Basic-Programmzeilen, die geändert werden sollen, müssen hier zum Beispiel mit einer Art EDIT-Befehl extra aufgerufen werden, um dann einzeln bearbeitet werden zu können.

Vom Basic-Interpreter oder vom Betriebssystem her beschränken sich die Funktionen des Editors im wesentlichen auf zwei Funktionen: Das Einlesen von Zeichen über die Tastatur und das Ausgeben von Zeichen auf dem Bildschirm. Hinzu kommen beim C128 noch einige Funktionen, die zum Beispiel Bildschirmfenster setzen und ähnliches.

Der Aufruf der Editor-Funktionen erfolgt über eine Sprungleiste. Unter einer Sprungleiste ist eine Reihe von JUMP-Befehlen zu verstehen, die ihrerseits zu den eigentlichen Arbeitsroutinen springen. Hier werden Sie sich fragen, wozu solch eine Sprungleiste nützlich ist. Auf den ersten Blick stellt sie doch lediglich eine gewisse Platzverschwendung und, gerade bei zeitkritischen Programmen, eine Verlängerung der Befehls-Ausführungszeit dar.

Der Grund ist eigentlich ganz einfach. Auch Betriebssysteme sind Programme, und für Programme gilt der schöne Satz: Ein Programm ist nie fehlerfrei; es sei denn, es ist veraltet.

Während der Programmierung eines doch recht umfangreichen Betriebssystems treten immer wieder Fehler zutage, die eine Änderung am Programm erfordern. Hiermit geht aber leider auch eine gewisse Verschiebung der Programmroutinen einher, womit ebenfalls alle Programme zu ändern wären, die auf das Betriebssystem oder, in diesem Fall den Editor, zugreifen.

Bei Verwendung einer Sprungleiste ist man dieser Arbeiten enthoben, da die Adresse der Funktion innerhalb der Sprungleiste immer gleich bleibt. Nur die JUMP-Befehle zeigen jetzt unter Umständen auf andere Adressen.

An dieser Stelle sollen die wichtigsten Routinen des Betriebssystem kurz erläutert werden, um aufzuzeigen, wie der Assembler-Programmierer diese Routinen sinnvoll verwenden kann.

Weiterhin wollen wir ein wenig unsere Trickkiste öffnen und – gerade am noch recht unbekannten C128 – ein paar Tricks und Kniffe zeigen. Hierzu gehört unserer Meinung nach aber auch dann und wann ein Hinweis auf das, was man einen guten Programmierstil nennt, zum Beispiel auch die Verwendung der oben erwähnten Sprungleisten, gerade auch in Ihren eigenen Programmen. Umfangreiche Programme haben die Eigenart, nie ganz fertig zu sein und gerade die Verwendung solcher, doch eigentlich ganz simpler, Hilfsmittel kann eine Menge Arbeit ersparen. Vor allem dann, wenn man nicht so komfortable Hilfsmittel wie einen Assembler zur Verfügung hat, der ein Programm, von sagen wir acht KByte, vollständig neu assembliert; natürlich unter Umrechnung aller absoluten Sprünge.

Die erwähnte Sprungleiste umfaßt 16 Funktionen, die teilweise vom Anwender verwendet werden können und die auch das restliche Betriebssystem verwendet.

Zunächst einmal eine tabellarische Auflistung der möglichen Funktionen:

Name	Adresse	Funktion
jpcint	$C000	Videochip/Editor initialisieren
jdsply	$C003	Ausgabe AC auf Schirm, Attribut in XR
jlp2	$C006	Zeichen aus Tastatur-Puffer nach AC
jloop5	$C009	Zeichen vom Schirm nach AC
jprint	$C00C	Schirm-Ausgabe AC
jscorg	$C00F	Schirm-Organisation lesen
jkey	$C012	Tastatur-Abfrage
jrep	$C015	Code aus Tast.-Tab. holen, REPEAT-Test
jplot	$C018	Cursor-Position lesen/schreiben
jcurs	$C01B	Cursor-Position im 80-Zeichen-BS setzen
jesc	$C01E	Bearbeitung der ESC-Funktionen
jkyset	$C021	Funktionstaste belegen
jrsirq	$C024	Raster-Interrupt bearbeiten
jint80	$C027	80-Zeichen-Schirm initialisieren
jswap	$C02A	Umschalten 40/80-Zeichen-Schirm
jwind	$C02D	Schirm-Fenster definieren

Die einzelnen Routinen werden im folgenden jeweils kurz besprochen, da sie einige der wichtigsten Routinen für die Kommunikation mit dem Computer von eigenen Programmen enthalten.

jpcint ($C000) Videochip/Editor initialisieren

Diese Routine setzt nach dem Einschalten des Computers beziehungsweise einem Drücken des RESET-Tasters die Grundeinstellung der Video-Controller und der für das Editorprogramm wichtigen Speicheradressen.

jdsply ($C003) Ausgabe AC auf Schirm, Attribut in XR

Hiermit kann ein beliebiges Zeichen direkt auf den Bildschirm gebracht werden, wobei das Zeichen im Akku stehen und bereits im Bildschirmcode vorliegen muß. Das X-Register muß das hierfür gültige Attribut-Byte enthalten, also die Farb-

information für den 40-Zeichen-Bildschirm, und für den 80-Zeichen-Bildschirm zusätzlich die Attribute für Unterstreichen, Blinken und so weiter.

Das Attribut-Byte hat folgenden Aufbau:

Bit	40-Zeichen-Schirm	80-Zeichen-Schirm
0-3	Farbinformation	Farbinformation
	0000 = Schwarz	0000 = Schwarz
	0001 = Weiß	0001 = Grau
	0010 = Rot	0010 = Blau
	0011 = Grün	0011 = Hellblau
	0100 = Violett	0100 = Dunkelgrün
	0101 = Dunkelgrün	0101 = Hellgrün
	0110 = Blau	0110 = Dunkelgrau
	0111 = Gelb	0111 = Grün
	1000 = Hellbraun	1000 = Rot
	1001 = Braun	1001 = Rosa
	1010 = Rosa	1010 = Hellbraun
	1011 = Dunkelgrau	1011 = Violett
	1100 = Grau	1100 = Braun
	1101 = Hellgrün	1101 = Gelb
	1110 = Hellblau	1110 = Hellgrau
	1111 = Hellgrau	1111 = Weiß
4	-	0 = Blinken aus
		1 = Blinken an
5	-	0 = Unterstreichen aus
		1 = Unterstreichen an
6	-	0 = Normale Darstellung
		1 = Inverse Darstellung
7	-	0 = Grafik/Großbuchstaben
		1 = Klein-/Großbuchstaben

Im 40-Zeichen-Modus werden die Farben, und damit das Attribut-Byte, wie beim C 64 dargestellt. Es werden also lediglich die untersten 4 Bit verwendet.

Im 80-Zeichen-Modus wird lediglich das Bit 6 ($40) nicht verwendet. Wie auch beim 40-Zeichen-Bildschirm werden die inversen Zeichen durch ein gesetztes oberstes Bit des Schirmcodes dargestellt ($00 bis $7F = normales Zeichen, $80 bis $FF = inverses Zeichen). Das Bit 7 (Grafikdarstellung) ist im Zusammenhang mit der <SHIFT+CBM>-Taste zu sehen, mit der bekanntlich von Großbuchstaben/Grafikzeichen auf Klein-/Großschreibung umgeschaltet werden kann. Im Gegensatz zum 40-Zeichen-Modus beziehungsweise zum C 64 ist es hier aber möglich, Zeichen aus beiden Zeichensätzen gleichzeitig auf dem Schirm zu sehen. Während im 40-Zeichen-Modus der Zeichensatz umgeschaltet wird, bestimmt hier lediglich das oberste Bit des Attribut-Bytes, welcher Zeichensatz für das einzelne Zeichen herangezogen werden soll. Hierdurch sind insgesamt 512 verschiedene Schriftzeichen gleichzeitig auf dem Bildschirm darstellbar, und zwar 128 Zeichen, diese in inverser Darstellung (=256) und das Ganze noch in zwei Zeichensätzen (=512).

Der Zeichensatz des VDC befindet sich im eigenen 16-KByte-RAM-Bereich des VDC und kann mit eigenen Routinen verändert werden. Hierdurch ist es zum Beispiel möglich, die dort enthaltenen inversen Zeichen durch eigene Zeichen zu ersetzen, und die Revers-Darstellung durch das Setzen des entsprechenden Attribut-Bits zu erreichen. Hierdurch läßt sich die Anzahl der darstellbaren Zeichen noch einmal verdoppeln, so daß insgesamt 1024 (!) verschiedene Zeichen gleichzeitig auf dem Schirm sichtbar sein können.

Die Routine »jdsply« schreibt das Zeichen an die aktuell gesetzte Schreibposition, die durch die Speicherstellen »pnt« ($EO/$E1) als Basisadresse und »pntr« ($SEC) als ffset markiert wird. Die zugehörige Adresse im Attributspeicher wird durch die Routine selbständig errechnet und befindet sich nach der Rückkehr in der Adresse »user« ($E2/$E3).

jlp2 $C006 Zeichen aus Tastaturpuffer nach AC

Während des Interrupts werden Tastendrücke zunächst nur registriert und in den dafür vorgesehenen Tastaturpuffer von zehn Zeichen eingelesen. Mit der Routine »jlp2« können Sie nun aus eigenen Programmen Zeichen von der Tastatur holen, wobei die Zeichen dann aus dem Tastaturpuffer geholt werden, sofern dort Zeichen vorhanden sind.

Nach dem Aufruf der Funktion enthält der Akkumulator das gelesene Zeichen oder, wenn kein Tastendruck vorliegt, eine Null. Der Inhalt des Tastaturpuffers wird durch die Routine um dieses eine Byte vermindert. Die Speicherstelle »ndx« ($D0) gibt dabei an, wieviel Zeichen sich gerade im Tastaturpuffer befinden.

Die Funktionstasten <F1> bis <F8> sowie auch die <HELP>-Taste und die <SHIFT+RUN/STOP>-Taste werden dabei etwas anders behandelt. Auf die Funktionstasten <F1> bis <F8> können mit dem KEY-Befehl Texte oder auch Befehle gelegt werden, die dann beim Drücken der entsprechenden Taste ausgeführt werden. Hierzu dient ein weiterer Zähler »kyndx« ($D1), der angibt, wieviel Zeichen auszugeben sind und das Indexfeld »keyidx« ($D2), das auf das aktuelle Zeichen aus der Key-Definition zeigt. Die Funktionstasten haben dabei Vorrang vor dem normalen Tastaturpuffer. Wenn zum Beispiel die Tasten <A>, <F1> und <B> betätigt werden, werden bei mehrfachem Aufruf von »jlp2« zunächst die einzelnen Zeichen des Tastaturstrings der Funktionstaste <F1>, sofern vorhanden, zugewiesen und anschließend daran das »A« und das »B« gelesen.

jloop5 $C009 Zeichen vom Schirm nach AC

Die Routine »jloop5« liest ein Zeichen aus dem Bildschirmspeicher im ASCII-Format. Gelesen wird wieder aus der Adresse, die durch die aktuelle Bildschirmposition »pnt« ($E0/$E1) und den Offset »pntr« ($EC) angegeben wird. Nach der Rückkehr enthält der Akkumulator das gelesene Zeichen und die Variable »tcolor« ($F2) das dazugehörende Attribut-Byte.

jprint $C00C Schirm-Ausgabe AC

Mit Hilfe dieser Routine ist es möglich, ein Zeichen auf den Bildschirm zu bringen, wobei hier das Zeichen, im Gegensatz zur Routine »jdsply«, im ASCII-Format übergeben werden muß. Es können hier alle Zeichen, auch Steuerzeichen wie Cursor-Bewegungen, Farbumschaltungen und so weiter verwendet werden. Diese Routine wird zum Beispiel auch vom Basic innerhalb des PRINT-Befehls aufgerufen.

jsorg $C00F Schirm-Organisation lesen

Beim C 128 ist es, im Gegensatz zum C 64, wie bei den großen CBM-Computern möglich, ein Bildschirmfenster zu setzen. Das bedeutet, daß die normalerweise eingestellte Bildschirmgröße von 25 Zeilen zu 40 beziehungsweise 80 Zeichen jederzeit vom Anwender geändert werden kann, und irgendwelche Schirm-Ausgaben, zumindestens mit PRINT-Befehlen, sich nur noch auf dieses Schirmfenster beziehen. Mit der Routine »jscorg« können Sie sich jederzeit einen Überblick verschaffen, wie die momentane Aufteilung des Bildschirms ist. Nach dem Aufruf der Routine enthält das X-Register die momentane Anzahl zulässiger Spalten (Zeichen pro Zeile), das Y-Register die maximal zulässige Anzahl Zeilen und der Akku die Zeilenbreite insgesamt, also entweder 40 oder 80.

jkey $C012 Tastatur-Abfrage

Diese Routine übernimmt die eigentliche Tastatur-Abfrage und das Bereitstellen der eingelesenen Zeichen im Tastaturpuffer. Für den Anwender ist sie unerheblich, da diese Abfrage im Interrupt selbsttätig vorgenommen wird.

jrep $C015 Code aus Tastaturtabelle holen, REPEAT-Test

Auch diese Routine ist für den Anwender uninteressant, da sie im Interrupt selbsttätig ausgeführt wird. Aufgrund der, in der Tastatur-Abfrage ermittelten, Tastennummer wird aus der zuständigen Tastaturtabelle der zugehörige ASCII-Wert ermittelt und geprüft, ob für diese Taste eine Wiederholung (Repeat) zulässig ist, sofern die Taste noch seit dem letzten Aufruf gedrückt ist.

jplot $C018 Cursor-Position lesen/schreiben

Hiermit ist es jederzeit möglich festzustellen, an welcher Position auf dem Bildschirm sich momentan der Cursor befindet, unabhängig davon, ob er gerade sichtbar ist oder nicht. Die Cursor-Position ist im übrigen die Position, an der das nächste mit PRINT beziehungsweise der Routine »jprint« auszugebende Zeichen erscheint.

Um die Cursor-Position zu lesen, muß zuvor das Carry-Flag mit dem Befehl »SEC« gesetzt werden. Nach dem Aufruf der Routine enthält das X-Register die aktuelle Zeilennummer der Cursor-Position und das Y-Register die aktuelle Position innerhalb der Zeile. Beide Werte sind immer relativ zur linken oberen Ecke des aktuellen Bildschirmfensters zu sehen.

Zum Schreiben der Cursor-Position muß vor dem Aufruf zunächst das Carry-Flag mit dem Befehl »CLC« gelöscht werden. Außerdem muß das X-Register die gewünschte Zeilennummer und das Y-Register die gewünschte Spalte enthalten; beide Werte wieder relativ zur linken oberen Ecke des Schirmfensters. Die erfolgreiche Ausführung des Befehls wird durch ein gelöschtes Carry-Flag angezeigt. Ist das Carry-Flag gesetzt, ist ein Fehler aufgetreten, also zum Beispiel eine Zeilennummer verwendet worden, die nicht in die aktuelle Fenstergröße paßt.

Die aktuelle Zeichenposition »pnt« ($E0/$E1), der Spaltenoffset »pntr« ($EC) sowie der Zeiger auf den Attributspeicher »user« ($E2/$E3) werden entsprechend neu errechnet.

jcurs $C01B Cursor-Position im 80-Zeichen-BS setzen

Im 80-Zeichen-Bildschirm wird der Cursor hardwaremäßig vom VDC erzeugt, also nicht wie beim 40-Zeichen-Bildschirm durch einfaches Invertieren des aktuellen Zeichens. Mit der Routine »jcurs« wird die aktuelle Cursor-Position an den VDC übergeben und dort auf die entsprechende Stelle im Bildschirmspeicher gesetzt. Die Adresse wird wieder durch die aktuelle Zeichenposition »pnt« ($E0/$E1) und den Spaltenoffset »pntr« ($EC) markiert, die selbstverständlich vorher auf die gewünschte Adresse zu setzen sind.

Das An- beziehungsweise Abschalten des Cursors kann leider nicht über eine Routine der Editor-Sprungleiste erreicht werden. Die entsprechenden Routinen heißen »crsron« ($CD6F) zum Einschalten und »crsrof« ($CD9F) zum Abschalten des Cursors. Beide Routinen wirken sich immer auf den gerade aktuellen Schirm, also wahlweise den 40-Zeichen- oder 80-Zeichen-Bildschirm, aus.

Der Cursor-Modus für den 40-Zeichen-Bildschirm wird durch die Adresse »blnon« ($0A26) festgelegt. Ist dort das Bit 6 ($40) gesetzt, wird ein fester Cursor erzeugt, ist das Bit nicht gesetzt, wird ein blinkender Cursor erzeugt.

Im 80-Zeichen-Modus sind noch ein paar Steuerungen mehr möglich. Hierzu dient die Adresse »curmod« ($0A2B), die bitweise die gewünschte Cursor-Form angibt. Die einzelnen Bits haben folgende Bedeutung, wobei nicht alle Möglichkeiten im C 128 genutzt, aber durchaus verwendet werden können:

Bit	Bedeutung
7	Nicht benutzt
6	Wenn das Bit "1" ist, blinkt der Cursor; ist das Bit "0", steht der Cursor fest
5	Dieses Bit arbeitet in Zusammenhang mit Bit 6: Ist Bit 6 = "0" und Bit 5 = "1", dann ist der Cursor abgeschaltet. Ist Bit 6 = "1", bestimmt dieses Bit die Blinkfrequenz: "0" = schnell "1" = langsam (Normalzustand)
4	Immer "0"
3	Sollte immer "0" sein; das Bit wird als höchstwertiges Bit zu den folgenden Bits 0 bis 2 betrachtet
0-2	Diese Bits geben an, an welcher Punktposition innerhalb des Zeichens der Cursor beginnen soll

Beim C 128 wird ein Zeichen aus acht Punktreihen zu jeweils acht Punkten, also insgesamt 64 Punkten, zusammengesetzt. Der Cursor wird durch Invertieren der entsprechenden vertikalen Reihen dargestellt, wobei durch die Bits 0 bis 2 gewählt wird, an welcher Position er beginnen soll. Sind alle Bits »0« beginnt der Cursor an der ersten Position; es wird also ein Block-Cursor dargestellt. Mit dem binären Wert 7 (alle drei Bit = »1«) beginnt der Cursor in der letzten Punktposition; es wird also ein Unterstrich-Cursor dargestellt. Es sind hier natürlich auch Zwischenwerte möglich, die allerdings recht seltsam aussehen können (halber Cursor und ähnliches).

Das Bit 3 stellt das oberste Bit dieser Punktkombination dar. Der Grund liegt darin, daß der VDC durch andere Programmierung Zeichen auch mit 16 Punktzeilen darstellen könnte. Die Auflösung ist dabei natürlich erheblich besser, und die Zeichen sind in sich viel geschlossener und schärfer. Leider sind Farbmonitore erschwinglicher Preisklasse hiermit überfordert, daher wurde darauf verzichtet. Diese Möglichkeit merkt man übrigens auch bei einer Analyse der im VDC-RAM abgelegten Zeichensätze. Im Zeichensatz-ROM belegen die beiden Zeichensätze zusammen nur 4 KByte; im VDC-RAM jedoch 8 KByte. Hinter jeder Zeichendefinition, die aus 8 Byte besteht, befindet sich hier jeweils eine weitere Gruppe von 8 Null-Byte, um die 16 möglichen Punktreihen zu belegen.

Hier noch einmal die sinnvollen Cursor-Formen in tabellarischer Übersicht:

Wert	Cursor-Form
$00	Block-Cursor, fest
$07	Unterstrich-Cursor, fest
$20	Cursor ausgeschaltet
$40	Block-Cursor, schnell blinkend
$47	Unterstrich-Cursor, schnell blinkend
$60	Block-Cursor, langsam blinkend
$67	Unterstrich-Cursor, langsam blinkend

jesc $C01E Bearbeitung der ESC-Funktionen

Der C 128 besitzt diverse Editor-Funktionen, die über eine ESCAPE-Sequenz aufgerufen werden. Hierzu wird zunächst <ESC> gedrückt und anschließend ein Buchstabe von <A> bis <Z> oder das <§>-Zeichen, beziehungsweise der <@> bei der ASCII-Tastatur. Ein nochmaliges <ESC> direkt hinter dem ersten schaltet die Funktion sofort wieder aus, falls <ESC> versehentlich gedrückt wurde. Diese Funktionen können auch über PRINT-Befehle beziehungsweise die oben erwähnte Routine »jprint« ausgeführt werden.

Die Routine »jesc« ruft die entsprechenden Editor-Funktionen direkt, ohne vorheriges <ESC>, auf. Hierzu ist vor dem Aufruf im Akkumulator der gewünschte Befehlsbuchstabe zu setzen.

Die möglichen Befehle in tabellarischer Auflistung:

Akkuinhalt	Funktion
§	Bildschirm ab Cursor-Position löschen
A	Autoinsert-Modus einschalten. Vor der Ausgabe eines Zeichens wird vorher mit »Insert« automatisch eine freie Stelle eingerichtet.
B	Rechte untere Fensterecke an aktueller Cursor-Position setzen
C	Autoinsert-Modus ausschalten
D	Aktuelle Zeile löschen. Der Rest des Bildschirms wird nach oben gerollt
E	Festen Cursor setzen
F	Blinkenden Cursor setzen
G	Glocke zulassen. Beim Betätigen von <CTRL+G> Glockensignal, desgleichen bei der Ausgabe von CHR$(7) auf dem Bildschirm.
H	Glocke abschalten. Ein Aufruf der Glocke wird ignoriert
I	Zeile an der Cursor-Position einfügen. Die folgenden Zeilen verschieben sich nach unten
J	Cursor an den Anfang der Zeile setzen
K	Cursor auf Zeilenende setzen
L	Bildschirm-Rollen einschalten. Wird bei PRINT-Befehlen der untere Rand erreicht, werden die Zeilen nach oben gerollt und eine Leerzeile am Schirmende eingefügt.

M	Bildschirm-Rollen ausschalten. Wird bei PRINT-Befehlen der untere Rand erreicht, wird die Ausgabe wieder am oberen Schirmrand fortgesetzt.
N	Negativ-Darstellung ausschalten (siehe »R«)
O	Einfüge-, Anführungszeichen- und Reverse-Modus (zum Beispiel nach Anführungszeichen) ausschalten
P	Zeile bis Cursor-Position löschen
Q	Zeile ab Cursor-Position löschen
R	Negativ-Darstellung einschalten. Der gesamte Bilschirminhalt wird invertiert, allerdings nicht der Bilschirmrahmen
S	Block-Cursor einschalten
T	Obere linke Fensterecke an aktuelle Cursor-Position setzen
U	Unterstrich-Cursor einschalten
V	Schirm um eine Zeile aufwärts rollen
W	Schirm um eine Zeile abwärts rollen
X	Umschalten von 40- auf 80-Zeichen-Modus und zurück
Y	Standard-Tabulatoren setzen
Z	Alle Tabulatoren löschen

jkyset $C021 Funktionstaste belegen

Hiermit können die Funktionstasten von Assemblerprogrammen aus mit Werten belegt werden. Hierzu sind zunächst einige Vorbereitungen zu treffen. In der Zeropage ist an drei aufeinander folgenden Adressen ein Zeiger auf den neuen, der Funktionstaste zuzuordnenden Text einzurichten. Anschließend ist in den Akkumulator die Adresse des ersten Bytes dieses Zeigers zu laden, in das X-Register die Nummer der Funktionstaste laut unten stehender Tabelle und in das Y-Register die Länge des Textes.

Die Funktionstaste <F1> soll zum Beispiel mit dem Text »Testbelegung« definiert werden, der in der Bank 1 an der Adresse $3000 steht. Hierzu wird zunächst einmal ein Zeiger in der Zeropage eingerichtet, sinnvollerweise an einer sonst nicht benötigten Stelle, zum Beispiel den Adressen $FA bis $FF.

Belegt wird also

$FA mit $00, dem Low-Byte der Adresse $3000
$FB mit $30, dem High-Byte der Adresse $3000
$FC mit $01, der Banknummer

Anschließend werden die Prozessor-Register geladen, und zwar:
AC mit $FA, der Zeigeradresse
XR mit $01, der Nummer der Funktionstaste
YR mit $0C, der Länge des Textes

Nun kann die Routine aufgerufen und die Funktionstaste belegt werden. Mögliche Funktionstasten sind die Tasten <F1> bis <F8>, die <HELP>-Taste und die <SHIFT-RUN/STOP>-Taste. Die anzugebende Nummer finden Sie in der folgenden Tabelle:

Nummer	Taste	Standardwert
1	<F1>	GRAPHIC
2	<F2>	DLOAD"
3	<F3>	DIRECTORY + CHR$(13)
4	<F4>	SCNCLR + CHR$(13)
5	<F5>	DSVAVE"
6	<F6>	RUN + CHR$(13)
7	<F7>	LIST + CHR$(13)
8	<F8>	MONITOR + CHR$(13)
9	<SHIFT+RUN/STOP>	DL"*" + CHR$(13) + RUN + CHR$(13)
10	<HELP>	HELP + CHR$(13)

Beachten Sie bitte beim Belegen der Funktionstasten, daß alle zehn Definitionen zusammen nicht mehr als 246 Zeichen umfassen dürfen.

jrsirq $C024 Raster-Interrupt bearbeiten

Diese Routine ist für den Anwender nicht von Belang, da sie im Interrupt automatisch aufgerufen wird. Sie dient zur Umschaltung von Grafik- und Textbildschirm, wenn im Basic ein Grafikmodus mit geteiltem Schirm eingestellt wurde. Beim Aufbau wird das Bild jeweils zeilenweise aufgebaut, wobei eine Zeile immer einer vertikalen Punktbreite entspricht. Der gesamte Bildschirm umfaßt theoretisch etwa 250 Zeilen inklusive des oberen und unteren Rahmens, wobei, je nach Monitor, einige Zeilen nicht sichtbar sind. Der VIC kann nun veranlaßt werden, bei Erreichen einer bestimmten Rasterzeile einen Interrupt auszulösen. Solche Interrupts werden von dieser Routine bearbeitet, die dann jeweils vom Grafikschirm auf den Textschirm umschalten und somit einen geteilten Bildschirm simulieren.

jint80 $C027 80-Zeichen-Schirm initialisieren

Mit dieser Routine wird der Zeichengenerator in das RAM des VDC kopiert. Wie schon erwähnt, wird beim Betätigen der <ASCII/DIN>-Taste der jeweils aktuelle Zeichensatz übertragen. Diese Funktion übernimmt die vorliegende Routine, die den Zeichengenerator von der Adresse $D000 in Bank 14 in das RAM des VDC kopiert.

jswap $C02A Umschalten 40/80-Zeichen-Schirm

Diese Funktion schaltet auf den jeweils anderen Bildschirm, also von 40 Zeichen auf 80 Zeichen oder umgekehrt, um. Sie entspricht vollständig der Funktion »ESC X« und wird von dieser auch aufgerufen.

jwind $C02D Bildschirmfenster definieren

Mit dieser Funktion können Bildschirmfenster gesetzt werden, wie auch schon mit den Funktionen »ESC B« beziehungsweise »ESC T«. Vor dem Aufruf ist im Akkumulator die gewünschte Zeilennummer absolut, das heißt also von 0 bis 24 und im X-Register die gewünschte Spalte, ebenfalls absolut, also von 0 bis 39 beziehungsweise 0 bis 79, anzugeben. Bei gesetzem Carry-Flag wird die rechte untere Ecke des Fensters gesetzt, bei gelöschtem Carry-Flag die linke obere Ecke.

Das Kernel

Das Kernel umfaßt alle Funktionen, die zum ordnungsgemäßen Betrieb des Computers notwendig sind, und ruft seinerseits auch den Editor auf. Im Kernel sind alle Routinen enthalten, um den Datenverkehr auf dem seriellen Bus zur Floppy oder einem Drucker zu regeln, sowie die gesamte Bedienung der RS232-Schnittstelle und des Kassettenrecorders. Weiterhin übernimmt es die Bearbeitung von Interrupts, also IRQ und NMI sowie die erstmalige Einstellung aller Systemparameter nach dem Einschalten oder dem Drücken des Reset-Tasters.

Auch das Kernel verfügt über eine Sprungleiste, die hier allerdings am Ende des Programms liegt und über die alle notwendigen Funktionen aufgerufen werden können. Der grundlegende Teil der Sprungleiste ist übrigens mit der des C64 identisch; sie ist nur um einige Funktionen erweitert worden, die beim C128 hinzugekommen sind.

Hier zunächst einmal eine tabellarische Auflistung der einzelnen Funktionen, und anschließend eine eingehende Besprechung der wichtigsten Routinen. Alle Routinen unterhalb der punktierten Linie sind übrigens identisch mit denen des C64.

kspio $FF47 Auf I/O schnelle Floppy umschalten

Mit dieser Funktion wird auf die schnelle Floppyroutine umgeschaltet, die allerdings nur mit den Floppylaufwerken 1570 und 1571 verwendet werden kann. Das Carry-Flag bestimmt, ob die Routine eingeschaltet (Carry gelöscht) oder ausgeschaltet (Carry gesetzt) werden soll.

kclsal $FF4A Alle Files von Gerät in AC schließen

Mit dieser Funktion können alle Daten gleichzeitig geschlossen werden, die auf der im Akku angegebenen Geräteadresse geöffnet sind. Ein gesetzes Carry-Flag nach der Rückkehr zeigt einen Fehler innerhalb der Routine an.

Name	Adresse	Funktion
kspio	$FF47	Auf I/O schnelle Floppy umschalten
kclsal	$FF4A	Alle Files von Gerät in AC schließen
kc64md	$FF4D	Auf C 64-Modus umschalten
kdmacl	$FF50	DMA-Anforderung ext. RAMs zulassen
kbotcl	$FF53	Boot-Programm von Diskette laden
kphenx	$FF56	Aufruf Kaltstartroutinen Funktionskarte
klkupl	$FF59	Geräte- & Sek.-Adr. über la (ac) suchen
klkups	$FF5C	Geräte-Adr. über Sek.-Adr. (yr) suchen
kswapr	$FF5F	Umschalten 40/80-Zeichen-BS
kdlchr	$FF62	80-Zeichen-BS initialisieren
kpfkey	$FF65	Funktionstaste belegen
kstbnk	$FF68	Bank für LOAD/SAVE/VERIFY setzen
kgtcfg	$FF6B	MMU-Wert zu Bank in xr nach ac
kjsrfr	$FF6E	JSR, Bank xr
kjmpfr	$FF71	JMP, Bank xr
kifetv	$FF74	LDA (FETVEC),y von Bank in xr
kistav	$FF77	STA (STAVEC),y in Bank in xr
kicmpv	$FF7A	CMP (CMPVEC),y mit Bank in xr
kprimm	$FF7D	Folgenden Text ausgeben und überspringen
kpcint	$FF81	Videochips/Editor initialisieren
kioini	$FF84	I/O-Bausteine initialisieren
kramts	$FF87	RAM, Kass.-Puffer u. Vectoren einrichten
kresto	$FF8A	Standard-I/O-Vektoren einrichten
kvecto	$FF8D	User-I/O-Vektoren einrichten
kstmsg	$FF90	Flag setzen Unterdrückung Meldungen
ksecnd	$FF93	Ausgabe Sek.-Adr. nach LISTEN auf IEC-Bus
ktsksa	$FF96	Ausgabe Sek.-Adr. nach TALK auf IEC-Bus
kmemtp	$FF99	höchste Speichergrenze lesen/schreiben
kmembt	$FF9C	unterste Speichergrenze lesen/schreiben
kscnky	$FF9F	Tastatur-Abfrage
ksetmo	$FFA2	Timeout-Flag IEC-Bus setzen
kacptr	$FFA5	Zeichen von IEC-Bus nach AC
kciout	$FFA8	Ausgabe ac auf IEC-Bus
kuntlk	$FFAB	Ausgabe UNTALK auf IEC-Bus
kunlsn	$FFAE	Ausgabe UNLISTEN auf IEC-Bus
klistn	$FFB1	Ausgabe LISTEN auf IEC-Bus
ktalk	$FB4	Ausgabe TALK auf IEC-Bus
kreads	$FFB7	Aktueller Status nach AC
kstlfs	$FFBA	la, fa, sa aus ac, xr, yr setzen
kstnam	$FFBD	fnlen, fnadr aus ac, xr, yr setzen
kopen	$FFCO	Logisches File öffnen
kclose	$FFC3	Logisches File schließen
kchkin	$FFC6	Eingabe-Kanal öffnen
kckout	$FFC9	Ausgabe-Kanal öffnen
kclrch	$FFCC	Ein/Ausgabe-Kanal schließen
kbasin	$FFCF	Zeichen akt. Kanal nach AC
kbsout	$FFD2	Ausgabe AC auf akt. Kanal
kloads	$FFD5	LOAD/VERIFY (AC) ab Adresse in xr/yr
ksaves	$FFD8	Save ab ac bis xr, yr
ksttim	$FFDB	System-Uhr setzen
krdtim	$FFDE	System-Uhr nach ac, xr, yr
kstop	$FFE1	Stop-Taste lesen
kgetin	$FFE4	Zeichen akt. Kanal nach AC
kclall	$FFE7	Filetabelle löschen, Vorgabe-I/O
kclock	$FFEA	System-Uhr aktualisieren
kscror	$FFED	BS-Organisation lesen
kplot	$FFFO	Cursor-Pos. lesen/schreiben
kiobas	$FFF3	Basisadresse I/O-Bausteine nach xr/yr

kc64md $FF4D Auf C 64-Modus umschalten

Diese Routine schaltet auf den C 64-Modus um, wobei ein Rücksprung von dort nur mit der Reset-Taste möglich ist. Diese Routine wird auch vom Basic-Befehl »GO64« aufgerufen.

kdmacl $FF50 DMA-Anforderung externer RAMs zulassen

Diese Routine dient zum DMA (Direct Memory Access = direkter Speicherzugriff) durch externe Geräte. Im Normalbetrieb des C 128 ist sie nicht von Belang.

kbotcl $FF53 Boot-Programm von Diskette laden

Mit dem Basic-Befehl »BOOT« kann zum Beispiel das CP/M-Betriebssystem geladen werden, in dem der Block 0 in der Spur 1 der Diskette gelesen wird, wo weitere Anweisungen wie Programmname, Startadresse und so weiter zu finden sind. Diese Routine ruft die entsprechende Funktion auf, wobei im Akkumulator die Laufwerknummern 0 oder 1 im ASCII-Code, also $30 oder $31, und im X-Register die Gerätenummer stehen müssen. Wenn der gelesene Sektor ein Bootsektor ist, wird das Programm ausgeführt, ansonsten wird der Aufruf ignoriert.

kphenx $FF56 Aufruf Kaltstartroutinen Funktionskarte

Diese Routine dient zum Aufruf der zusätzlichen Funktionskarte, die im normalen C 128 nicht enthalten ist. Wird die Routine hier aufgerufen, wird ein normaler »Boot« von Gerät 8, Laufwerk 0, durchgeführt.

klkupl $FF59 Geräte- und Sekundäradresse über la (ac) suchen

Zu einer vorgegebenen logischen Dateinummer im Akkumulator werden die zugehörige Gerätenummer und die Sekundäradresse in der Dateitabelle gesucht. Das Carry-Flag zeigt an, ob ein entsprechender Eintrag in der Dateitabelle gefunden wurde. Ist das Carry-Flag gesetzt, ist diese Datei nicht geöffnet. Ist das Carry-Flag gelöscht, enthält das X-Register die Geräteadresse, das Y-Register die Sekundäradresse und der Akkumulator die logische Dateinummer.

klkups $FF5C Geräteadresse über Sekundäradresse (yr) suchen

Zu einer vorgegebenen Sekundäradresse im Y-Register werden die zugehörige logische Dateinummer und die Sekundäradresse gelesen. Das Carry-Flag zeigt an, ob ein entsprechender Eintrag in der Dateitabelle gefunden wurde. Ist das Carry-Flag gesetzt, ist diese Datei nicht geöffnet. Ist das Carry-Flag gelöscht, enthält das X-Register die Geräteadresse, das Y-Register die Sekundäradresse und der Akkumulator die logische Dateinummer.

kswapr $FF5F Umschalten 40/80-Zeichen-Bildschirm

Diese Routine ruft die unter den Editor-Funktionen beschriebene Routine »jswap« auf.

kdlchr $FF62 80-Zeichen-Bildschirm initialisieren

Diese Routine ruft die unter den Editor-Funktionen beschriebene Routine »jint80« auf.

kpfkey $FF65 Funktionstaste belegen

Diese Routine ruft die unter den Editor-Funktionen beschriebene Routine »ikyset« auf.

kstbnk $FF68 Bank für LOAD/SAVE/VERIFY setzen

Beim Eröffnen von Diskettendateien müssen in der Regel Dateinamen angegeben werden, deren Adressen in der Speicherstelle »fnadr« ($BB/$BC) zu finden sind. Beim C 128 muß darüber hinaus noch die Bank angegeben werden, auf die sich diese Adresse bezieht. Die Bank wird mit dieser Funktion gesetzt, wobei das X-Register die entsprechende Bit-Kombination für das MMU-Register und der Akkumulator die Banknummer enthalten muß.

kgtcfg $FF6B MMU-Wert zu Bank in XR nach AC

Zu jeder RAM- oder ROM-Bank des C 128 gehört eine entsprechende Bit-Kombination, die in das Konfigurationsregister der MMU zu schreiben ist, um die gewünschte Speicherverteilung zu erhalten. Hierzu existiert im Kernel-ROM eine entsprechende Tabelle. Diese Routine sucht zu einer gegebenen Banknummer im X-Register die entsprechende Kombination aus der Tabelle und stellt sie im Akkumulator zur Verfügung.

kjsrfr $FF6E JSR, Bank xr

Diese Routine ruft ein Unterprogramm auf, das sich in einer beliebigen Speicherbank befinden kann. Hierzu muß das X-Register die Banknummer enthalten, und die Speicherstellen »areg« und folgende die entsprechenden gewünschten Registerinhalte beim Eintritt in das Unterprogramm. Die Logik entspricht hier einem JSR-Befehl.

kjmpfr $FF71 JMP, Bank xr

Diese Routine ruft ein Programm auf, das sich in einer beliebigen Speicherbank befinden kann. Hierzu muß das X-Register die Banknummer enthalten, und die Speicherstellen

»areg« und folgende die entsprechenden gewünschten Registerinhalte beim Eintritt in das Programm. Die Logik entspricht hier einem JMP-Befehl.

kifetv $FF74 LDA (FETVEC),y von Bank in xr

Mit Hilfe dieser Routine kann ein Byte aus einer anderen Speicherbank als der, in der das aktuelle Programm gerade läuft, gelesen werden. Hierzu muß das Y-Register die Banknummer und der Akkumulator die Adresse eines Adressenpaars aus der Zeropage enthalten, in der die zu lesende Adresse steht. Die Funtkion entspricht somit in etwa einem »LDA (Adresse),y«-Befehl.

kistav $FF77 STA (STAVEC),y in Bank in xr

Mit Hilfe dieser Routine kann ein Byte in eine andere Speicherbank als die, in der das aktuelle Progrmm gerade läuft, geschrieben werden. Hierzu muß das X-Register die Banknummer und die Speicherstelle »stashx« ($02B9) die Anfangsadresse eines Registerpaars aus der Zeropage enthalten, in der die zu schreibende Adresse steht. Die Funktion entspricht somit in etwa einem »STA (Adresse),y«-Befehl.

kicmpv $FF7A CMP (CMPVEC),y mit Bank in xr

Mit Hilfe dieser Routine kann ein Byte im Akkumulator mit einem Byte aus einer anderen Speicherbank als der, in der das aktuelle Programm gerade läuft, verglichen werden. Hierzu muß das X-Register die Banknummer enthalten, und die Speicherstelle »cmparx« ($02CD) die Anfangsadresse eines Registerpaars aus der Zeropage, in der die zu vergleichende Adresse steht. Die Funktion entspricht somit in etwa einem »CMP (Adresse),y«-Befehl.

kprimm $FF7D Folgenden Text ausgeben und überspringen

Mit Hilfe dieser Routine kann ein beliebiger Text, der direkt auf den JSR-Befehl folgt, auf dem Bildschirm ausgegeben werden. Der Text kann beliebig lang sein und muß mit »$00« beendet werden. Anschließend wird die Verarbeitung mit dem auf den Text folgenden Befehl fortgesetzt. Diese Routine ist im übrigen ein Musterbeispiel für trickreiche Programmierung und ist somit durchaus eine genaue Analyse wert.

kpcint $FF81 Videochips/Editor initialisieren

Diese Routine ruft die unter den Editor-Funktionen besprochene Routine »jpcint« auf.

kioini $FF84 I/O-Bausteine initialisieren

Mit dem Aufruf dieser Routine werden alle Ein-/Ausgabe-Bausteine auf ihre Standardwerte nach dem Einschalten des Rechners gesetzt. Wird ein Anwender-Programm von anderen Programmen aufgerufen und herrscht Unklarheit über die aktuelle Einstellung der Peripherie-Bausteine, so kann hiermit ein definierter Zustand hergestellt werden.

kramts $FF87 RAM, Kassettenpuffer und Vektoren einrichten

Mit dieser Routine werden die Zeiger auf die Ein-/Ausgabe-Puffer neu eingerichtet und die Speichergrenzen auf die Standardwerte festgelegt.

kresto $FF8A Standard-I/O-Vektoren einrichten

Diese Routine stellt die normalen I/O-Vektoren, zum Beispiel für IRQ, Öffnen einer Datei und so weiter wieder her. Sie kann aufgerufen werden, wenn diese Vektoren ganz oder teilweise verändert wurden, um den Grundzustand wiederherzustellen.

kvecto $FF8D User-I/O-Vektoren einrichten

Hiermit können die I/O-Vektoren auf eigene Adressen gesetzt werden. Hierzu muß im X-Register das Low-Byte und im Y-Register das High-Byte der Vektorentabelle gesetzt werden. Die Vektorentabelle selbst muß alle 16 möglichen Vektoren enthalten, um Fehler zu vermeiden. Sie baut sich aus jeweils einem Wort (2 Byte) in der Reihenfolge Low-Byte, High-Byte auf, die auf die Adressen der aufzurufenden Routinen zeigen. Die Original-Vektortabelle »vectss« ($E073) kann hierfür teilweise kopiert werden.

kstmg $FF90 Flag setzen Unterdrückung Meldungen

Bei der Abarbeitung von Systemroutinen können Fehler auftreten, wie zum Beispiel ein nicht angegebener Dateiname und ähnliches. Normalerweise würde dann auf dem Bildschirm eine entsprechende Fehlermeldung erscheinen. Diese Routine setzt ein Flag, mit dem die Ausgabe dieser Meldung verhindert (Akku = $40) oder ermöglicht (Akku = $00) werden kann. Zusätzlich kann durch ein gesetztes Bit 7 ($80 beziehungsweise $C0) verhindert werden, daß bei Operationen, wie zum Beispiel dem Laden von Programmteilen, der Text »searching for...« oder auch »press play...« auf dem Bildschirm erscheint.

ksecnd $FF93 Ausgabe Sekundäradresse nach LISTEN auf IEC-Bus

Hiermit wird die gewünschte Sekundäradresse nach einem Listen-Befehl auf dem IEC-Bus ausgegeben, um dem Peripheriegerät mitzuteilen, über welche Sekundäradresse die folgenden Daten abzulegen sind. Die gewünschte Sekundäradresse muß sich beim Aufruf im Akkumulator befinden.

Hierzu kurz noch eine Anmerkung: Die gesamte Steuerung des IEC-Bus ist recht komplex und kann schnell durcheinander kommen, wenn die korrekte Reihenfolge der Listen-, Unlisten-, Talk- und sonstigen Befehle nicht eingehalten wird. Noch nicht so erfahrene Programmierer sollten daher auf die eigene Anwendung der IEC-Routinen verzichten, und besser mit den weiter unten beschriebenen Funktionen »kchkin«, »kckout«, »kgetin« und »kbsout« arbeiten. Bei der Verwendung dieser Routinen übernimmt das Betriebssystem die gesamte Steuerung des IEC-Bus; allerdings sind diese Routinen insgesamt etwas langsamer als die direkte Steuerung.

ktksa $FF96 Ausgabe Sekundäradresse nach TALK auf IEC-Bus

Hiermit wird die gewünschte Sekundäradresse nach einem Talk-Befehl auf dem IEC-Bus ausgegeben, um dem Peripheriegerät mitzuteilen, über welche Sekundäradresse die folgenden Daten zu senden sind. Die gewünschte Sekundäradresse muß sich beim Aufruf im Akkumulator befinden. Ansonsten gilt auch hier der unter »ksecnd« gegebene Hinweis.

kmemtp $FF99 Höchste Speichergrenze lesen/schreiben

Diese Routine liest oder schreibt wahlweise die höchste Speicheradresse, die vom Benutzerprogramm (und hier ist auch der Basic-Interpreter ein Benutzer!) verwendet werden kann. Über das Carry-Flag wird gesteuert, ob die Adresse geschrieben (Carry nicht gesetzt) oder gelesen (Carry gesetzt) werden soll. Beim Setzen muß im X-Register das Low-Byte und im Y-Register das High-Byte der Adresse stehen; beim Lesen enthalten die angegebenen Register die momentan gültigen Werte.

kmembt $FF9C Unterste Speichergrenze lesen/schreiben

Diese Routine liest oder schreibt wahlweise die unterste Speicheradresse, die vom Benutzerprogramm verwendet werden kann. Über das Carry-Flag wird gesteuert, ob die Adresse geschrieben (Carry nicht gesetzt) oder gelesen (Carry gesetzt) werden soll. Beim Setzen muß im X-Register das Low-Byte und im Y-Register das High-Byte der Adresse stehen; beim Lesen enthalten die angegebenen Register die momentan gültigen Werte.

kscnky $FF9F Tastatur-Abfrage

Hiermit wird die unter den Editor-Funktionen besprochene Routine »jkey« aufgerufen.

ksetmo $FFA2 Timeout-Flag IEC-Bus setzen

Diese Funktion wird vom System nicht verwendet. Die entsprechende Speicherstelle für das Verhindern des IEC-Timeout, wie zum Beispiel bei den großen CBM-Rechnern, ist zwar vorhanden, wird aber nicht verwendet. Somit dürfte die Routine nur als Platzhalter anzusehen sein, um ein Verschieben der restlichen Adressen zu verhindern.

kacptr $FFA5 Zeichen von IEC-Bus nach AC

Diese Routine liest ein Zeichen direkt vom IEC-Bus in den Akkumulator. Zuvor müssen die entsprechenden Talkroutinen aufgerufen werden. Hierfür gilt ansonsten auch der unter »ksecnd« geschriebene Hinweis.

kciout $FFA8 Ausgabe ac auf IEC-Bus

Das Zeichen im Akkumulator wird direkt auf dem IEC-Bus ausgegeben. Zuvor müssen die entsprechenden Listenroutinen aufgerufen werden. Hierfür gilt ansonsten auch der unter »ksecnd« gegebene Hinweis.

kuntlk $FFAB Ausgabe UNTALK auf IEC-Bus

Auf dem IEC-Bus wird ein »Untalk« ausgegeben, also die Aufforderung, das Senden von Daten einzustellen. Hierfür gilt ansonsten auch der unter »ksecnd« gegebene Hinweis.

klistn $FFB1 Ausgabe LISTEN auf IEC-Bus

Hiermit wird auf dem IEC-Bus ein »Listen« ausgegeben, also die Kennzeichnung, daß jetzt Daten-Bytes folgen. Hierfür gilt ansonsten auch der unter »ksecnd« gegebene Hinweis.

ktalk $FFB4 Ausgabe TALK auf IEC-Bus

Hiermit wird auf dem IEC-Bus ein »Talk« ausgegeben, also die Kennzeichnung, daß jetzt Daten zu senden sind. Hierfür gilt ansonsten auch der unter »ksecnd« gegebene Hinweis.

kreads $FFB7 Aktueller Status (RS232 beziehungsweise Syst.) nach AC

Hiermit kann der aktuelle Status nach einer Ein-/Ausgabeoperation abgefragt werden. Der Status enthält eine Null, wenn die Operation fehlerfrei abgeschlossen wurde, ansonsten wird, bitweise verschlüsselt, ein Hinweis auf die Art des Fehlers gegeben. Die einzelnen Bits haben, sofern sie auf »1« gesetzt sind, folgende Bedeutung:

Bit	Bedeutung
0	Timeout (Zeitüberschreitung) beim Schreiben
1	Timeout (Zeitüberschreitung) beim Lesen
2	Zu kurzer Datenblock (Nur Kassette)
3	Zu langer Datenblock (Nur Kassette)
4	Schwerer Fehler, nicht weiter spezifiziert
5	Prüfsummenfehler
6	Dateiende, kein Fehler, nur ein Hinweis
7	Angesprochenes Gerät nicht angeschlossen oder Bandende (nur Kassette)

kstlfs $FFBA la, fa, sa aus ac, xr, yr setzen

Die aktuellen Parameter für eine zu öffnende Datei können hiermit gesetzt werden. Der Akkumulator muß die logische Dateinummer, das X-Register die Gerätenummer und das Y-Register die Sekundäradresse enthalten. Zusätzlich ist im Anschluß an diese Funktion unter Umständen noch der Dateiname mit der im folgenden beschriebenen Funktion »setnam« anzugeben.

kstnam $FFBD fnlen, fnadr aus ac, xr, yr setzen

Hiermit können die Parameter für einen Dateinamen beim Eröffnen einer Datei gesetzt werden. Der Akkumulator muß die Länge des Dateinamens und, sofern die Länge nicht null – also kein Dateiname vorhanden – ist, das X-Register das Low-Byte der Dateinamenadresse und das Y-Register das High-Byte enthalten. Zusätzlich ist noch mit der Routine »setbnk« (siehe dort) die Banknummer zu definieren, in der der Dateiname steht. Falls nicht schon geschehen, werden außerdem mit der oben beschriebenen Routine »kstlfs« die Parameter für Dateinummer, Geräte- und Sekundäradresse definiert.

kopen $FFC0 Logisches File öffnen

Eine zuvor mit den oben beschriebenen Parametern versehene Datei kann mit dieser Routine geöffnet werden. Nach der Operation zeigt das Carry-Flag an, ob die Funktion erfolgreich (Carry nicht gesetzt) oder fehlerhaft (Carry gesetzt) war.

kclose $FFC3 Logisches File schließen

Eine zuvor geöffnete Datei kann mit dieser Funktion wieder geschlossen werden. Der Akkumulator muß hierzu die logische Dateinummer enthalten. Nach der Operation zeigt das Carry-Flag an, ob das Eröffnen erfolgreich (Carry nicht gesetzt) oder fehlerhaft (Carry gesetzt) war.

kchkin $FFC6 Eingabe-Kanal öffnen

Diese Routine dient zum Einlesen von Daten von beliebigen Geräten. Zuvor ist eine entsprechende Datei zu öffnen, wobei hier jedes Gerät verwendet werden kann, das als Eingabegerät zugelassen ist, also IEC-Bus (Gerätenummern 4 bis 15), Kassette (#1), RS232-Schnittstelle (#2) und Bildschirm (#0).

Beim Aufruf der Routine muß das X-Register die logische Dateinummer enthalten. Alle folgenden Aufrufe zum Beispiel der Routine »kgetin« (siehe dort) lesen jeweils ein Zeichen vom angesprochenen Gerät, bis mit der Routine »kclrch« die Eingabe wieder auf die Tastatur zurückgestellt wird.

kckout $FFC9 Ausgabe-Kanal öffnen

Diese Routine dient zum Ausgeben von Daten auf beliebige Geräte. Zuvor ist eine entsprechende Datei zu öffnen. Dabei kann jedes Gerät verwendet werden, das als Ausgabegerät zugelassen ist, also IEC-Bus (Gerätenummern 4 bis 15), Kassette (#1), RS232-Schnittstelle (#2) und Bildschirm (#3).

Beim Aufruf der Routine muß das X-Register die logische Dateinummer enthalten. Alle folgenden Aufrufe zum Beispiel der Routine »kbsout« (siehe dort) schreiben jeweils ein Zeichen auf das angesprochene Gerät, bis mit der Routine »kclrch« die Ausgabe wieder auf den Bildschirm zurückgestellt wird.

kclrch $FFCC Ein/Ausgabe-Kanal schließen

Diese Routine stellt nach erfolgter Datenein- oder -ausgabe wieder auf den Bildschirm als Ausgabe- und die Tastatur als Eingabegerät zurück.

kbasin $FFCF Zeichen vom aktiven Kanal nach AC

Hiermit wird ein Zeichen vom aktiven Eingabe-Kanal gelesen. Normalerweise wird ein Zeichen aus der aktuellen Bildschirmposition gelesen, es sei denn, mit »kchkin« ist zuvor eine Datei als Eingabe-Kanal aktiviert worden.

kbsout $FFD2 Ausgabe AC auf aktiven Kanal

Hiermit wird ein Zeichen auf dem aktiven Ausgabe-Kanal ausgegeben. Normalerweise wird ein Zeichen an der aktuellen Bildschirmposition ausgegeben, es sei denn, mit »kckout« wurde zuvor eine Datei als Ausgabe-Kanal aktiviert.

kloads $FFD5 Load/verify (AC) ab Adresse in xr/yr

Hiermit kann ein Programm geladen beziehungsweise mit einem im Speicher stehenden Programm verglichen werden. Hierzu sind zunächst die Parameter für den Dateinamen zu setzen, und dann die Routine aufzurufen. Im Akkumulator muß angegeben werden, ob ein »Load« (Akku = $00) oder ein »Verify« (Akku <> $00) durchgeführt werden soll. Der Lade- oder Vergleichsvorgang beginnt ab der in der Datei angegebenen Startadresse, sofern die Sekundäradresse null ist. Ist die Sekundäradresse ungleich null, so wird die Datei an die Adresse geladen, die im X-Register (Low-Byte) und im Y-Register (High-Byte) angegeben ist.

ksaves $FFD8 Save ab ac bis xr, yr

Mit dieser Routine kann ein im Speicher stehendes Programm in einer Datei gesichert werden. Hierzu sind zunächst die Parameter für den Dateinamen zu setzen, und dann die Routine aufzurufen. Im Akkumulator muß die Adresse eines Zeropage-Registerpaares angegeben werden, das die Startadresse im Speicher enthält. Die Endadresse wird beim Aufruf mitgeteilt, wobei das X-Register das Low-Byte und das Y-Register das High-Byte enthalten muß.

ksttim $FFDB Systemuhr setzen

Das Betriebssystem des C128 stellt eine Systemuhr zur Verfügung, die jeweils im Interrupt fortgeschrieben wird. Mit

dieser Routine kann die Uhr auf einen neuen Wert gesetzt werden, wobei die Uhrzeit in $^1/_{60}$-Sekunden anzugeben ist. Eine Sekunde entspricht somit dem Wert 60, eine Minute dem Wert 3600 und eine Stunde dem Wert 216000. Die Uhrzeit ist aus diesen Werten zu errechnen, wobei zum Beispiel 10 Uhr, 5 Minuten und 20 Sekunden dem Wert 2179200 entspricht. Diese Zahl ist ins binäre Datenformat umzurechnen und in drei Byte der Routine zu übergeben, wobei der Akkumulator das höchstwertige Byte enthält. In unserem Beispiel enthalten die Register folgende Werte:
Akku = 33, XR = 64 und YR = 128,
denn (33 * 65536) + (64 * 256) + 128 = 2179200.

krdtim $FFDE Systemuhr nach ac, xr, yr

Die vom C 128 verwaltete Systemuhr kann mit dieser Routine gelesen werden, wobei die Uhrzeit als Anzahl der seit Null Uhr vergangenen $^1/_{60}$-Sekunden zu interpretieren ist. Eine Beschreibung finden Sie beim vorhergehenden Befehl. Die Register enthalten die Werte in der dort angegebenen Reihenfolge.

kstop $FFE1 <RUN/STOP>-Taste lesen

Durch diese Routine kann festgestellt werden, ob die <RUN/STOP>-Taste gedrückt wurde. Nach dem Aufruf der Funktion ist das Zero-Flag entsprechend gesetzt, das heißt ein BEQ-Befehl direkt nach der Routine verzweigt, wenn die <RUN/STOP>-Taste gedrückt wurde. Zu beachten ist, daß die Routine auch gleichzeitig – natürlich nur bei gedrückter <RUN/STOP>-Taste – die Standard-Ein/Ausgabe-Kanäle Tastatur und Bildschirm wieder herstellt.

kgetin $FFE4 Zeichen vom aktiven Kanal nach AC

Die Routine liest ein Zeichen vom aktiven Kanal in den Akkumulator, also im Normalfall von der Tastatur, es sei denn, mit »kchkin« ist der Eingabe-Kanal zuvor auf ein anderes Gerät umgeleitet worden.

kclall $FFE7 Filetabelle löschen, Vorgabe-I/O

Diese Routine setzt zunächst wieder die Standard-Ein/Ausgabe-Kanäle und schließt gleichzeitig alle eventuell offenen Dateien aller Gerätenummern.

kclock $FFEA Systemuhr aktualisieren

Diese Routine, die vom Interrupt aufgerufen wird, erhöht die Systemuhr um eine $^1/_{60}$-Sekunde. Sie sollte nur benutzt werden, wenn Sie eine eigene Interrupt-Routine und die Systemuhr in Ihrem Programm verwenden. Es muß allerdings sichergestellt sein, daß nur der periodisch ausgelöste System-Interrupt und nicht etwa Interrupts durch Ein-/Ausgabe-Bausteine diese Routine aufrufen; ansonsten ist mit großen Abweichungen in der Genauigkeit der Uhr zu rechnen.

kscror $FFED Bildschirm-Organisation lesen

Diese Routine ruft die unter den Editor-Funktionen beschriebene Funktion »jscorg« auf.

kplot $FF0 Cursor-Position lesen/schreiben

Diese Routine ruft die unter den Editor-Funktionen beschriebene Funktion »jplot« auf.

kiobas $FFF3 Basis-Adresse I/O-Bausteine nach xr/yr

Hiermit erhalten Sie nach dem Aufruf die Basis-Adresse der Ein-/Ausgabe-Bausteine ($D000) im X-Register (Low-Byte) und Y-Register (High-Byte).

Damit ist unsere Reise durch das Betriebssystem des C 128 abgeschlossen. Bleibt nur noch, Ihnen viel Erfolg beim kreativen Programmieren mit Ihrem Computer zu wünschen. Auch wenn beim ersten Anlauf nicht alles so funktioniert, wie Sie es sich vorstellen, lassen Sie den Kopf nicht hängen. Es ist noch kein Meister vom Himmel gefallen. (ah)

Der vorstehende Artikel ist ein Auszug aus dem Buch:
C 128 ROM-Listing, erschienen bei Markt&Technik, Hans-Pinsel-Straße 2, 8013 Haar, Best.-Nr.: ISBN 3-89090-221-9, Preis: 49 Mark

Tips & Tricks zum C 128

Vielseitige Hilfen und Erweiterungen erhalten Sie regelmäßig in den Tips & Tricks. Neben vielen nützlichen Programmen sind diesmal ein komfortabler RENUMBER-Befehl, eine 80-Zeichen-Hardcopy mit vielen Finessen und ein leistungsfähiger DATA-Wandler dabei.

Dem C 128 fehlen noch viele nützliche Befehle und Erweiterungen. Die Rubrik »Tips & Tricks« gibt Ihnen Hilfen, neue Basic-Befehle und allerlei nützliche Programme an die Hand, die von Profis extra für Sie geschrieben wurden, um Ihnen das Arbeiten zu erleichtern.

LIST-Hilfe für Basic-Programme

Dieses Programm (Listing 1) ermöglicht ein LISTen von Programmzeilen mit Hilfe der <CRSR>-Tasten. So wird ein unmittelbares Editieren von Programmzeilen möglich, die dauernde Eingabe von zum Beispiel »LIST 80-134« überflüssig. Das Programm belegt etwas mehr als 400 Byte im Kassetten- und RS232-Eingabepuffer.

Das Programm wird mit

```
BLOAD"LIST-HILFE"
```

geladen und durch

```
BANK 15:SYS 2816
```

gestartet. Das Programm belegt nun die Taste <F1> mit dem Text:

```
BANK 15:SYS 2816
```

so daß bei mehrmaliger Initialisierung nur <F1> gedrückt werden muß.

Die Routine läßt sich mit <RUN/STOP+RESTORE> abschalten.

Eine gleichzeitige Benutzung des Kassettenpuffers oder der RS232-Schnittstelle ist nicht möglich, da das Programm die Puffer ab $0B00 belegt. Durch Änderung weniger Adressen ist es jedoch leicht möglich, die Routine in einen anderen Speicherbereich der BANK 15 zu verschieben.

Während der normalen Programmier-Tätigkeit macht sich das Programm nicht bemerkbar. Wird der Cursor jedoch über den oberen oder unteren Bildschirmrand hinausbewegt, tritt das Programm in Aktion.

Steht auf dem sichtbaren Bildschirm eine Zeilennummer am linken Rand, wird entweder am unteren Bildschirmende die Folgezeile oder am oberen Bildschirmrand die vorhergehende Basic-Zeile angezeigt.

Steht auf dem Bildschirm keine Zeilennummer oder ist die erste beziehungsweise letzte Zeile gelistet, scrollt der Bildschirm auf- oder abwärts.

Wurde das Scrollen durch <ESC> + <M> unterbunden, wird die entsprechende Basic-Zeile ohne Zeilenverschiebung oben beziehungsweise unten in den Bildschirm geschrieben, soweit die Zeile hineinpaßt.

Findet der Editor keine Zeilennummer, springt der Cursor bei <ESC> + <M> von der ersten in die letzte Bildschirmzeile und umgekehrt (auch dies ist eine Erweiterung der normalen Funktion).

```
10 RESTORE : PRINT "{CTRL+N,CLR}" TAB(20) "{R
   VSON} L I S T - H I L F E "
20 BANK 15: S=0: FOR I=2816 TO 3231: READ A$
   : POKE I,DEC(A$): S=S+DEC(A$): NEXT
30 IF S<>48382 THEN PRINT "{3DOWN,CTRL+O,2SP
   ACE,RVSON}FEHLER IN DEN DATAS !!!": END
40 BSAVE "LIST-HILFE",ON B15,P2816 TO P3231
2816 DATA A9,80,A2,0B,8D,3C,03,8E,3D,03,A9,8
     F,A2,0C,85,FC
2832 DATA 86,FD,A2,00,8E,FF,03,E8,A0,0F,A9,F
     C,4C,65,FF,18
2848 DATA C9,3A,B0,05,E9,2F,38,E9,D0,60,A2,0
     0,86,0A,86,16
2864 DATA 86,17,E6,0A,E9,2F,85,09,A5,17,85,2
     4,A5,16,0A,26
2880 DATA 24,0A,26,24,65,16,85,16,A5,24,65,1
     7,85,17,06,16
2896 DATA 26,17,A5,16,65,09,85,16,90,02,E6,1
     7,20,54,C8,20
2912 DATA 58,CB,85,FA,18,20,20,0B,90,C8,60,E
     A,EA,EA,A5,EB
2928 DATA A4,EC,85,FB,84,FC,60,A5,FB,A4,FC,8
     4,EC,4C,38,C9
2944 DATA 85,FE,A6,F4,D0,18,AE,FF,03,D0,7C,C
     0,07,D0,09,A5
2960 DATA D3,29,01,F0,0B,4C,19,0C,C0,53,F0,F
     9,C0,54,D0,67
2976 DATA 18,A6,EB,E4,E4,90,60,EE,FF,03,20,6
     E,0B,20,9F,CD
2992 DATA 24,F8,30,03,20,BC,CA,20,B1,CB,20,5
     8,CB,20,1F,0B
3008 DATA 90,0D,18,A6,E5,E4,EB,B0,45,20,70,C
     8,4C,B7,0B,18
3024 DATA 20,2A,0B,E6,16,D0,02,E6,17,20,64,5
     0,B0,02,F0,1D
3040 DATA A0,02,20,EC,42,85,16,C8,20,EC,42,8
     5,17,A5,FB,A4
3056 DATA E6,84,EC,20,38,C9,A5,17,A6,16,20,2
     3,51,A9,00,85
3072 DATA FE,20,77,0B,4E,FF,03,A4,D4,A5,FE,4
     C,AD,C6,24,F8
3088 DATA 10,04,A6,E5,86,FB,4C,FD,0B,18,A6,E
     5,E4,EB,D0,E7
3104 DATA EE,FF,03,20,9F,CD,24,F8,30,03,20,C
     A,CA,20,6E,0B
3120 DATA 20,50,C1,20,58,CB,20,1F,0B,90,0D,1
     8,A6,EB,E4,E4
3136 DATA B0,42,20,77,C3,4C,33,0C,20,2A,0B,2
     0,64,50,A5,61
3152 DATA A6,62,C5,2D,D0,04,E4,2E,F0,A3,85,6
     3,86,64,C6,62
3168 DATA A0,00,C8,F0,98,20,EC,42,C5,63,D0,F
     6,C8,20,EC,42
3184 DATA 88,C5,64,D0,ED,18,98,65,61,85,61,A
     9,00,65,62,85
3200 DATA 62,4C,E0,0B,24,F8,10,04,A6,E4,86,F
     B,4C,FD,0B,42
3216 DATA 41,4E,4B,31,35,3A,53,59,53,32,38,3
     1,36,0D,00,00
```

Listing 1. Komfortableres LISTen eines Basic-Programms

```
. 00b00  a9 80     lda #$80     Initialisierungsroutine
. 00b02  a2 0b     ldx #$0b     Verbiegung des Vektors "Tastendruck speichern"
. 00b04  8d 3c 03  sta $033c
. 00b07  8e 3d 03  stx $033d
. 00b0a  a9 8f     lda #$8f     Belegung von Fí mit "bank15:sys2816"
. 00b0c  a2 0c     ldx #$0c          zur Initialisierung nach Run/Stop - Restore
. 00b0e  85 fc     sta $fc
. 00b10  86 fd     stx $fd
. 00b12  a2 00     ldx #$00
. 00b14  8e ff 03  stx $03ff
. 00b17  e8        inx
. 00b18  a0 0f     ldy #$0f
. 00b1a  a9 fc     lda #$fc
. 00b1c  4c 65 ff  jmp $ff65

. 00b1f  18        clc          Überprüfung des Akku auf Ziffer
. 00b20  c9 3a     cmp #$3a
. 00b22  b0 05     bcs $0b29
. 00b24  e9 2f     sbc #$2f
. 00b26  38        sec          C-Flag wird gesetzt, wenn keine Ziffer
. 00b27  e9 d0     sbc #$d0
. 00b29  60        rts

. 00b2a  a2 00     ldx #$00     Integerzahl vom Bildschirm holen (Cursor-
. 00b2c  86 0a     stx $0a           position) nach $16/ $17
. 00b2e  86 16     stx $16      ähnlich der Routine im Basic-Interpreter ab
. 00b30  86 17     stx $17           $f50a0
. 00b32  e6 0a     inc $0a
. 00b34  e9 2f     sbc #$2f
. 00b36  85 09     sta $09
. 00b38  a5 17     lda $17
. 00b3a  85 24     sta $24
. 00b3c  a5 16     lda $16
. 00b3e  0a        asl
. 00b3f  26 24     rol $24
. 00b41  0a        asl
. 00b42  26 24     rol $24
. 00b44  65 16     adc $16
. 00b46  85 16     sta $16
. 00b48  a5 24     lda $24
. 00b4a  65 17     adc $17
. 00b4c  85 17     sta $17
. 00b4e  06 16     asl $16
. 00b50  26 17     rol $17
. 00b52  a5 16     lda $16
. 00b54  65 09     adc $09
. 00b56  85 16     sta $16
. 00b58  90 02     bcc $0b5c
. 00b5a  e6 17     inc $17
. 00b5c  20 54 c8  jsr $c854    Cursor rechts
. 00b5f  20 58 cb  jsr $cb58    Zeichen unter Cursor in Akku holen
. 00b62  85 fa     sta $fa
. 00b64  18        clc
. 00b65  20 20 0b  jsr $0b20    Akku auf Ziffer prüfen
. 00b68  90 c8     bcc $0b32    die Routine, wird wiederholt, solange Ziffern
. 00b6a  60        rts               aufeinanderfolgen
. 00b6b  ea        nop
. 00b6c  ea        nop
. 00b6d  ea        nop

. 00b6e  a5 eb     lda $eb      die Cursorpostion wird in $fb/ $fc zwischen-
. 00b70  a4 ec     ldy $ec           gespeichert
. 00b72  85 fb     sta $fb
. 00b74  84 fc     sty $fc
. 00b76  60        rts

. 00b77  a5 fb     lda $fb      der Cursor wird auf die Ausgangsposition gesetzt
. 00b79  a4 fc     ldy $fc
. 00b7b  84 ec     sty $ec
. 00b7d  4c 38 c9  jmp $c938
                                die gedrückte Taste wird überprüft
. 00b80  85 fe     sta $fe      die gedrückte Taste wird zwischengespeichert
. 00b82  a6 f4     ldx $f4
. 00b84  d0 18     bne $0b9e    springt, wenn Quote-Modus angeschaltet ist
. 00b86  ae ff 03  ldx $03ff    Zeiger auf "Unterprogramm ist aktiv"
. 00b89  d0 7c     bne $0c07    springt, wenn Unterprogramm noch aktiv
. 00b8b  c0 07     cpy #$07     Code für untere Cursortaste
. 00b8d  d0 09     bne $0b98
. 00b8f  a5 d3     lda $d3      Speicher für Shift-Taste
. 00b91  29 01     and #$01
. 00b93  f0 0b     beq $0ba0
. 00b95  4c 19 0c  jmp $0c19    "Cursor up" ist gedrückt
. 00b98  c0 53     cpy #$53     Code für "Cursor up"
. 00b9a  f0 f9     beq $0b95
. 00b9c  c0 54     cpy #$54     Code für "Cursor down"
. 00b9e  d0 67     bne $0c07
. 00ba0  18        clc          Hauptprogramm "Cursor down"
. 00ba1  a6 eb     ldx $eb
. 00ba3  e4 e4     cpx $e4
. 00ba5  90 60     bcc $0c07    springt, wenn Cursor nicht unten im Fenster
. 00ba7  ee ff 03  inc $03ff    Setzt Hilfszeiger
. 00baa  20 6e 0b  jsr $0b6e    Cursorposition speichern
. 00bad  20 9f cd  jsr $cd9f    Cursor ausschalten
. 00bb0  24 f8     bit $f8
. 00bb2  30 03     bmi $0bb7
. 00bb4  20 bc ca  jsr $cabc    "Scroll up", falls scrollen erlaubt
. 00bb7  20 b1 cb  jsr $cbb1    Cursor an den Anfang der Zeile
. 00bba  20 58 cb  jsr $cb58    Zeichen vom Bildschirm holen
. 00bbd  20 1f 0b  jsr $0b1f    Zeichen = Ziffer ?
. 00bc0  90 0d     bcc $0bcf    springen bei Ziffer
. 00bc2  18        clc
. 00bc3  a6 e5     ldx $e5      Cursor schon in erster Fensterzeile ?
. 00bc5  e4 eb     cpx $eb
. 00bc7  b0 45     bcs $0c0e    ja, dann zum normalen Programm springen
. 00bc9  20 70 c8  jsr $c870    Cursosr up
. 00bcc  4c b7 0b  jmp $0bb7
                                Ziffer am linken Bildschirmrand gefunden
. 00bcf  18        clc
. 00bd0  20 2a 0b  jsr $0b2a    gefundene Zahl lesen, nach Integer wandeln und
. 00bd3  e6 16     inc $16        in $16/ $17 ablegen
. 00bd5  d0 02     bne $0bd9    Zahl um 1 erhöhen
. 00bd7  e6 17     inc $17
. 00bd9  20 64 50  jsr $5064    Anfang der Basic-Zeile suchen, oder Anfang der
. 00bdc  b0 02     bcs $0be0    nächsten Zeile, falls sie nicht exisitiert
. 00bde  f0 1d     beq $0bfd    letzte Basiczeile überschritten, dann Ende
. 00be0  a0 02     ldy #$02
. 00be2  20 ec 42  jsr $42ec    Zeilennummer der gefundenen Basic-Zeile in
. 00be5  85 16     sta $16           $16/ $17 ablegen
. 00be7  c8        iny
. 00be8  20 ec 42  jsr $42ec
. 00beb  85 17     sta $17
. 00bed  a5 fb     lda $fb
. 00bef  a4 e6     ldy $e6      Cursor unten links auf den Bildschirm setzen
. 00bf1  84 ec     sty $ec
. 00bf3  20 38 c9  jsr $c938
. 00bf6  a5 17     lda $17
. 00bf8  a6 16     ldx $16
. 00bfa  20 23 51  jsr $5123    Basic-Zeile listen
. 00bfd  a9 00     lda #$00
. 00bff  85 fe     sta $fe
. 00c01  20 77 0b  jsr $0b77    alte Cursorpositon holen
. 00c04  4e ff 03  lsr $03ff    Hilfszeiger löschen
. 00c07  a4 d4     ldy $d4
. 00c09  a5 fe     lda $fe
. 00c0b  4c ad c6  jmp $c6ad    in normale Routine springen (Tastencode zwischen-
                                     speichern)
```

Listing 2. Quelltext der LIST-Routine

```
. 00c0e  24 f8     bit $f8     Cursor an den oberen Rand setzen, falls scrollen
. 00c10  10 04     bpl $0c16        verboten
. 00c12  a6 e5     ldx $e5
. 00c14  86 fb     stx $fb
. 00c16  4c fd 0b  jmp $0bfd
                               Hauptprogramm "Cursor up"
. 00c19  18        clc
. 00c1a  a6 e5     ldx $e5
. 00c1c  e4 eb     cpx $eb
. 00c1e  d0 e7     bne $0c07   springt, wenn Cursor nicht unten im Fenster
. 00c20  ee ff 03  inc $03ff   Setzt Hilfszeiger
. 00c23  20 9f cd  jsr $cd9f   Cursor ausschalten
. 00c26  24 f8     bit $f8
. 00c28  30 03     bmi $0c2d
. 00c2a  20 ca ca  jsr $caca   "Scroll down", falls scrollen erlaubt
. 00c2d  20 6e 0b  jsr $0b6e   Cursorposition speichern
. 00c30  20 50 c1  jsr $c150   "Cursor home"
. 00c33  20 58 cb  jsr $cb58   Zeichen vom Bildschirm holen
. 00c36  20 1f 0b  jsr $0b1f   Zeichen = Ziffer ?
. 00c39  90 0d     bcc $0c48   springen, bei Ziffer
. 00c3b  18        clc
. 00c3c  a6 eb     ldx $eb     Cursor schon in letzter Fensterzeile ?
. 00c3e  e4 e4     cpx $e4
. 00c40  b0 42     bcs $0c84   ja, dann zum normalen Programm springen
. 00c42  20 77 c3  jsr $c377   Cursosr down
. 00c45  4c 33 0c  jmp $0c33
                               Ziffer am linken Bildschirmrand gefunden
. 00c48  20 2a 0b  jsr $0b2a   gefundene Zahl in $16/ $17 ablegen
. 00c4b  20 64 50  jsr $5064   Anfang der Basic-Zeile suchen
. 00c4e  a5 61     lda $61     ist Basic-Zeile die erste Zeile ?
. 00c50  a6 62     ldx $62
. 00c52  c5 2d     cmp $2d
. 00c54  d0 04     bne $0c5a
. 00c56  e4 2e     cpx $2e
. 00c58  f0 a3     beq $0bfd   ja, dann Ende
. 00c5a  85 63     sta $63     Zeilenanfang zwischenspeichern
. 00c5c  86 64     stx $64
. 00c5e  c6 62     dec $62     Anfang der vorhergehenden Zeile suchen
. 00c60  a0 00     ldy #$00
. 00c62  c8        iny
. 00c63  f0 98     beq $0bfd   nicht gefunden, dann Ende
. 00c65  20 ec 42  jsr $42ec   Zeichen aus Bank 0 holen, Adresse in $61/ $62
. 00c68  c5 63     cmp $63
. 00c6a  d0 f6     bne $0c62
. 00c6c  c8        iny
. 00c6d  20 ec 42  jsr $42ec
. 00c70  88        dey
. 00c71  c5 64     cmp $64
. 00c73  d0 ed     bne $0c62
. 00c75  18        clc         Anfang der vorhergehenden Zeile gefunden
. 00c76  98        tya
. 00c77  65 61     adc $61     Zeilenanfang berechnen und in $61/ $62 ablegen
. 00c79  85 61     sta $61
. 00c7b  a9 00     lda #$00
. 00c7d  65 62     adc $62
. 00c7f  85 62     sta $62
. 00c81  4c e0 0b  jmp $0be0   gefundene Basic-Zeile listen

. 00c84  24 f8     bit $f8     Cursor an den unteren Rand setzen, falls
. 00c86  10 04     bpl $0c8c        scrollen verboten
. 00c88  a6 e4     ldx $e4
. 00c8a  86 fb     stx $fb
. 00c8c  4c fd 0b  jmp $0bfd
```

Listing 2. Quelltext der LIST-Routine (Schluß)

Das Programm funktioniert auf dem 40- und 80-Zeichen-Bildschirm sowie bei jeder beliebigen Fenstergröße.

Funktion der Routine

Die Startroutine bei $0B00 (2816) verbiegt den Vektor »TASTENDRUCK SPEICHERN« ($33C/$33D), die der Systeminterrupt alle $^{1}/_{60}$ Sekunde aufruft. Wird keine Cursortaste erkannt, verzweigt die Routine in die normale Abfrage. Andernfalls versucht das Programm festzustellen,

```
Name : blanker 64mode       c000 c03f
-------------------------------------
c000 : 78 a9 11 a0 c0 8d 14 03  74
c008 : 8c 15 03 a9 1e 85 b6 58  ae
c010 : 60 a6 b6 ca 86 b6 30 03  af
c018 : 4c 31 ea a9 1e 85 b6 a5  21
c020 : cb c9 1c d0 f3 a9 04 2c  e6
c028 : 8d 02 f0 ec ad 11 d0 49  c9
c030 : 10 8d 11 d0 ad 30 d0 49  97
c038 : 01 8d 30 d0 4c 31 ea ad  7b
```

Listing 3. »Blanker 64mode« (bitte mit dem MSE im C64-Modus eingeben)

ob eine Zeilennummer auf dem Bildschirm steht und ob es eine Folgezeile oder vorhergehende Zeile im Basic-Programm gibt (die genaue Funktion dieser Abfrage entnehmen Sie bitte dem kommentierten Quellcode aus Listing 2).

Der Zeiger $3FF wird bei Programmbeginn als Flag gesetzt. Wenn er gesetzt ist, wird das Programm nicht erneut angesprungen. Dies ist notwendig, da das LISTen einer Basic-Zeile häufig länger als $^{1}/_{60}$ Sekunde dauert. Die Routine würde sich dann immer wieder selbst aufrufen, was zu einer Endlosschleife oder Programmabsturz führt.

(Ralf Pfeiffer/dm)

C64-Modus mit 2 MHz

»Blanker C64Mode« (Listing 3) wird mit dem MSE im C64-Modus eingegeben und durch SYS 49152 gestartet. Wenn Sie dann <CTRL+B> kurze Zeit gedrückt halten, schaltet sich der FAST-Mode (2-MHz-Betrieb) ein. Dies macht sich durch ein Abschalten des Bildschirms bemerkbar. In diesem Zustand läuft der C128 in beiden Betriebsarten (C64 und C128) doppelt so schnell, womit man Compiler oder langwierige Berechnungen in Basic beschleunigen kann (etwa das Einlesen von DATA-Werten oder das Zeichnen hochauflösender Grafik).

Nochmaliges Drücken von <CTRL+B> schaltet wieder auf normale Geschwindigkeit. Durch <RUN/STOP+RESTORE> wird der Blanker abgeschaltet (Neustart durch SYS 49152).

Noch zwei wichtige Hinweise: BLANKER läuft nur auf dem C128 im C64-Modus. Außerdem darf der FAST-Modus bei

```
100  -; *** BLANKER C64-MODE ***
110  -; *** ================ ***
120  -;
200  -.EQ IRQVEC = $0314 ; IRQ-VEKTOR
210  -.EQ TASTE  =   $CB ; AKTUELLER TASTENCODE
220  -.EQ ALTIRQ = $EA31 ; ADRESSE DER ALTEN IRQ-ROUTINE
230  -.EQ BLANK1 = $D011 ; VIC-REGISTER #17
240  -.EQ BLANK2 = $D030 ; VIC-REGISTER #48
250  -.EQ CTRLFL = $028D ; FLAG FUER SHIFT,C=,CTRL
260  -.EQ ZAEHL  =   $B6 ; ZAEHLER
270  -;
280  -.BA $C000 ; AB 49152 ABLEGEN
290  -;
300  -;
310  -; INITIALISIERUNG DER NEUEN IRQ-ROUTINE
320  -;
330  -          SEI              ; INTERRUPT ABSCHALTEN
340  -;
350  -          LDA #<(NEUIRQ)   ; IRQ-VEKTOR
360  -          LDY #>(NEUIRQ)   ; AUF NEUE
370  -          STA IRQVEC       ; IRQ-ROUTINE
380  -          STY IRQVEC+1     ; STELLEN
390  -;
400  -          LDA #30
410  -          STA ZAEHL        ; ZAEHLER SETZEN
400  -;
410  -          CLI              ; INTERRUPT WIEDER EINSCHALTEN
420  -          RTS              ; ENDE DER INITIALISIERUNG
430  -;
440  -NEUIRQ    LDX ZAEHL
450  -          DEX
460  -          STX ZAEHL        ; ZAEHLER DEKREMENTIEREN
470  -          BMI PRUEF
480  -ENDEIRQ   JMP ALTIRQ       ; WEITER WIE BEI ALTEM IRQ
485  -;
490  -PRUEF     LDA #30
500  -          STA ZAEHL        ; ZAEHLER WIEDER SETZEN
505  -;
510  -          LDA TASTE
520  -          CMP #28          ; AUF "B" PRUEFEN
530. -          BNE ENDEIRQ      ; ENDE WENN <> "B"
535  -;
540  -          LDA #4           ; BIT FUER "CTRL" GESETZT
550  -          BIT CTRLFL       ; AUF CTRL-TASTE PRUEFEN
560  -          BEQ ENDEIRQ      ; NICHT CTRL GEDRUECKT, DANN ENDE
565  -;
570  -          LDA BLANK1
580  -          EOR #16          ; BLANK-BIT FLIPPEN
590  -          STA BLANK1       ; UND ABSPEICHERN
595  -;
600  -          LDA BLANK2
610  -          EOR #1           ; FAST-BIT INVERTIEREN
620  -          STA BLANK2       ; UND ABSPEICHERN
625  -;
630  -          JMP ALTIRQ       ; WEITER BEIM ALTEN IRQ
```

Listing 4. Kommentierter Quelltext von »Blanker 64mode«

```
Name : 10er-tast            c000 c088
-------------------------------------
c000 : 78 a9 0d a0 c0 8d 14 03  73
c008 : 8c 15 03 58 60 78 a9 ff  5b
c010 : 8d 00 dc a9 00 8d 03 dc  3c
c018 : 8d 2f d0 ad 01 dc c9 ff  45
c020 : f0 47 a2 02 a9 fe a0 07  c0
c028 : 8d 2f d0 ad 01 dc cd 01  67
c030 : dc d0 f8 6a 90 0f 88 10  c3
c038 : fa ad 2f d0 29 07 2a ca  f8
c040 : 10 e4 4c 69 c0 18 8a 0a  0e
c048 : 0a 0a 85 ff 98 65 ff aa  c3
c050 : bd 70 c0 c5 fe f0 05 85  c5
c058 : fe 20 35 eb a9 7f 8d 00  fe
c060 : dc a9 ff 8d 2f d0 4c 31  cf
c068 : ea a9 00 85 fe 4c 5c c0  1d
c070 : 13 1d 9d 11 91 2c 30 8d  f2
c078 : 33 39 36 0d 0a 2d 2b 1b  64
c080 : 31 37 34 32 09 35 38 3f  3a
```

Listing 5. Der 10er-Block läßt sich auch im C64-Modus sinnvoll nutzen (bitte mit dem MSE im C64-Modus eingeben)

Ansteuerung von Peripheriegeräten nicht eingeschaltet sein, da dies zu I/O-Fehlern führt.

Assembler-Freaks können aus dem kommentierten Assembler-Listing (Listing 4) interessante Informationen entnehmen. (Florian Müller/dm)

10er-Block auch im C64-Modus

Wer einen C128 besitzt und diesen im C64-Modus benutzt, dem wird bestimmt schon aufgefallen sein, daß der 10er-Block und die zusätzlichen Cursortasten nicht ansprechbar sind. Das hier abgedruckte Maschinenprogramm (Listing 5, bitte mit dem MSE im C64-Modus eingeben) behebt dieses kleine Übel.

Der Video-Chip im C128 unterscheidet sich von seinem Vorgänger unter anderem dadurch, daß er einen 3-Bit-Port besitzt, ähnlich dem in der CPU oder in den CIAs. Dieser Port hat im C128 die Aufgabe, die zusätzlichen Ziffern-, Cursor- und Steuertasten abzufragen.

Die im C64-Modus nutzbare Tastatur ist über je acht Spalten- und Zeilenleitungen mit den beiden Ports des CIA 1 verbunden. Die fehlenden drei Zeilenleitungen für die Zusatztasten werden am VIC angeschlossen. Aus Kompatibilitätsgründen ist diese Tastatur im C64-Modus nicht aktiv, denn das Betriebssystem müßte ja diesen VIC-Port unterstützen. Genau hier greift 10ER-TAST ein.

Der Interrupt-Vektor wird auf die neue Routine verbogen und die normale Tastatur vorerst »abgeklemmt«. Nun legt das Programm alle drei Tastaturzeilen im VIC auf LOW. Ist eine der Zusatztasten gedrückt, so erscheint an CIA 1, Port B, etwas anderes als $FF. Ist dies der Fall, werden nacheinander alle drei Zeilen einzeln auf LOW gelegt und überprüft, ob diese Zeile angesprochen wurde.

Ist die Zeile mit der gedrückten Taste gefunden, wird noch ermittelt, in welcher Spalte die Taste liegt. Aus den Werten für Zeile und Spalte errechnet sich die Position des zugehörigen ASCII-Zeichens in der Tastaturtabelle. Dieser Code wird nun in den Tastaturpuffer gebracht und erscheint danach auf dem Bildschirm.

Der Start des Programms erfolgt durch SYS 49152. Sofort ist der gesamte Zehnerblock im C64-Modus nutzbar. (Achim Kalwa/dm)

Variablen-Übersicht

Mit dem Programm »XLV 3.6« (Listing 6) werden sämtliche Variablen, die ein Programm verwendet, sortiert auf dem Drucker ausgegeben. Zu jeder Variable werden sämtliche Zeilennummern, in denen die Variable vorkommt, mit ausgegeben.

Die Unterteilung der Variablen erfolgt in die vier Gruppen Systemvariable und Konstanten, numerische Gleitkomma-Variable, numerische Ganzzahl-Variable und Zeichenketten-Variable.

Für jede Gruppe können je 51 Variable und für jede Variable je 127 Zeilennummern verarbeitet werden. Zum Anschluß eignet sich jeder beliebige Drucker (auf die Verwendung druckerspezifischer Steuerkommandos wurde verzichtet).

Bedienung

Das Programm ist durch

```
RUN"XLV*"
```

zu starten. Mit »$« können alle Programme auf der Diskette angezeigt werden.

Gewünschten Programmnamen eingeben. (Bei Programmnamen mit mehr als 14 Zeichen bitte mit »*« abkürzen, da sonst ein STRING TOO LONG-Error auftritt.)

Sofern der Drucker noch nicht eingeschaltet ist, wird dies

```
10 P$="XLV 3.6": REM 09.07.86 * R. VOGEL * C
   H-8887 MELS
20 TRAP 830: DCLOSE : COLOR 0,1: COLOR 4,1:
   PRINT "{2HOME,CLR}";: E$=CHR$(27): CR$=CH
   R$(13)
30 IF PEEK(215)>0 THEN FAST : CHAR 1,18,0,E$
   +"T"
40 T0$="{5SPACE}*****************************
   ****************************************
   **** "
50 PRINT "{WHITE}VARIABLEN UEBERSICHT{4SPACE
   }"P$CR$ MID$(T0$,6,31)CR$"{PURPLE}'$' = D
   IRECTORY{2SPACE}'E'=ENDE{DOWN}"
60 OPEN 3,0: DIM TA$(52,3): DIM T$(3): GOTO
   120
70 REM #### SUB READ ####
80 GET #1,X$: S%=ST: X%=ASC(X$): RETURN
90 PRINT CR$"{3UP}"E$"Q{CTRL+G}'"F$"'IST KEI
   N BASIC PGM{DOWN}": GOTO 120
100 PRINT CR$"{3UP}"E$"Q{CTRL+G}PGM '"F$"' N
    ICHT GEFUNDEN{DOWN}"
110 REM #### START PGM ####
120 DCLOSE : PRINT "PGM-NAME ODER '$': ";: I
    NPUT#3,F$: IF F$="E" THEN 800
130 IF F$="$" THEN PRINT "{WHITE}": CATALOG
    "*=P"
135 IF F$="$" THEN PRINT "{PURPLE}'RET' = WE
    ITER";: GET KEY Y$: CLR : GOTO 10
140 DOPEN #1,(F$)+",P": IF DS>0 THEN 100
150 GOSUB 80: IF X%<>1 THEN 90
160 GOSUB 80: IF X%<>28 THEN 90: ELSE E%=1:
    M%=1
170 PRINT CR$"{DOWN}BITTE GEDULD"
180 REM #### LOOP PGM LESEN ####
190 DO UNTIL S%>0: GOSUB 80: IF E% THEN 290
200 IF R% AND X%>0 THEN 330
210 IF Q% AND X%<>34 AND X%>0 THEN 330
220 IF X%=34 AND Q%=0 THEN Q%=1: P%=1: M%=0:
     GOTO 260
230 IF X%=34 THEN Q%=0: M%=1: GOTO 330
240 IF X$=":" THEN M%=0: PRINT "{LIG.GREEN}:
    ";: GOTO 260
250 IF(X%<255 AND X%>127) OR X%<32 THEN : M%
    =0: IF X%=143 THEN R%=1: P%=1: PRINT "{L
    IG.RED}REM";
260 GOSUB 350: IF P% THEN P%=0: GOTO 330
270 IF X$=":" OR(X%<255 AND X%>127) OR X%<32
     THEN M%=1
280 IF X%>0 THEN 330
290 E%=E%+1: ON E% GOTO 330,330,330,300,310
300 Z0$=CHR$(X%): W%=X%: GOTO 330
310 Z1$=CHR$(X%): W%=W%+256*X%: E%=0: Q%=0:
    R%=0
320 CHAR 1,0,8,E$+"D": CHAR 1,0,22,"{LIG.BLU
    E}": PRINT USING "#####";W%;: PRINT " ";
330 LOOP : DCLOSE : GOTO 600
340 REM #### VARIABLEN SUCHEN ####
350 IF M% THEN Z%=Z%+1: ELSE GOTO 460
360 IF Z%=1 AND((X$=>"A" AND X$=<"Z") OR X%=
    255) THEN Z$=X$: L%=1: RETURN
370 IF Z%=1 THEN Z%=0: RETURN
380 IF X$="(" THEN Z$=Z$+X$: GOTO 460
390 IF X$="%" OR X$="$" THEN Z$=Z$+X$: L%=2:
     A%=1: RETURN
400 IF(X$>="0" AND X$<="9") OR(X$>="A" AND X
    $<="Z") THEN 420
410 GOTO 460
420 IF A% THEN GOSUB 460: GOTO 350
430 IF L%=1 THEN Z$=Z$+X$: L%=2
440 RETURN
450 REM #### BUFFER SCHREIBEN ####
460 IF Z%=0 THEN RETURN
470 VA$=":{4SPACE}": MID$(VA$,2)=Z$: Z%=0: A
    %=0: ZL$=Z1$+Z0$
480 IF INSTR(":DS{2SPACE}:DS$ :EL{2SPACE}:ER
    {2SPACE}:↑{3SPACE}:ST{2SPACE}:TI{2SPACE}
    :TI$ ",VA$)>0 THEN
    PRINT "{YELLOW}*";: V%=0: GOTO 510
```

Listing 6. »XLV 3.6« – leistungsfähiger Variablen-Dump

```
490 IF INSTR(VA$,"$")>0 THEN PRINT "{YELLOW}
    $";: V%=3: GOTO 510
500 IF INSTR(VA$,"%")>0 THEN PRINT "{YELLOW}
    %";: V%=2: ELSE PRINT "{YELLOW}#";: V%=1
510 T%=INSTR(T$(V%),VA$): IF T%=0 THEN T$(V%
    )=T$(V%)+VA$: GOTO 510
520 I%=(T%-1)/5: IF RIGHT$(TA$(I%,V%),2)=ZL$
     THEN 540
530 TA$(I%,V%)=TA$(I%,V%)+ZL$
540 RETURN
550 REM #### DRUCKER NICHT READY ####
560 CLOSE 4: CHAR 1,0,8,E$+"D"+E$+"D"
570 CHAR 1,0,21,"{PURPLE}BITTE DRUCKER EINSC
    HALTEN UND"+CR$+"'RET' DRUECKEN.{2SPACE}
    (E=ENDE)"
580 GET KEY Y$: IF Y$="E" THEN 790
590 REM #### BUFFER DRUCKEN ####
600 OPEN 4,4: PRINT#4,"{5SPACE}PROGRAMMNAME
    :{3SPACE}"F$: PRINT#4,T0$
610 IF ST<>0 THEN 560: ELSE PRINT#4
620 FOR V=0 TO 3: IF T$(V)="" THEN 780
630 PRINT#4,"{5SPACE}";
640 IF V=0 THEN PRINT#4,"SYSTEMVARIABLEN UND
     KONSTANTEN";
650 IF V=1 THEN PRINT#4,"NUMERISCHE GLEITKOM
    MA VARIABLEN";
660 IF V=2 THEN PRINT#4,"NUMERISCHE GANZZAHL
     VARIABLEN";
670 IF V=3 THEN PRINT#4,"ZEICHENKETTEN VARIA
    BLEN";
680 PRINT#4," :": PRINT#4,T0$
690 FOR I=0 TO LEN(T$(V))/5-1
700 Y$="↑↑↑↑": FOR Y=0 TO LEN(T$(V))/5-1
710 IF Y$>MID$(T$(V),Y*5+2,4) THEN Y$=MID$(T
    $(V),Y*5+2,4): Y%=Y
720 NEXT Y: MID$(T$(V),Y%*5+2,4)="↑↑↑↑": PRI
    NT#4,Y$" -";
730 FOR J=0 TO LEN(TA$(Y%,V))/2-1
740 IF INT(J/12)=J/12 AND J>0 THEN PRINT#4:
    PRINT#4,"{5SPACE}-";
750 PRINT#4,USING "######";(ASC(MID$(TA$(Y%,
    V),J*2+1,1))*256)+ASC(MID$(TA$(Y%,V),J*2
    +2,1));
760 NEXT J: PRINT#4: PRINT#4
770 NEXT I: PRINT#4: PRINT#4
780 NEXT V: CLOSE 4
790 PRINT CR$"{PURPLE}FERTIG! 'RET'=ENDE{2SP
    ACE}'R'=RESTART": GET KEY Y$: IF Y$="R"
    THEN CLR : GOTO 10
800 FF$="MENU": DOPEN #2,(FF$): DCLOSE : IF
    DS=0 THEN RUN "MENU"
810 PRINT "{CLR}";: END
820 REM #### ERROR ####
830 PRINT "{CLR,PURPLE}*** ERROR *** " ERR$(
    ER)" IN LINIE "EL;
840 HELP : PRINT
850 PRINT "'RET'=RETRY (RESUME), SONST = ABB
    RUCH ";: GET KEY Y$
860 IF Y$=CHR$
```

Listing 6. Dieses Programm dient der Ausgabe aller in einem Basic-Programm verwendeten Variablen auf einem Drucker (Schluß)

nun angezeigt. Ist der Drucker bereit, so werden die Variablen sortiert ausgedruckt. (R. Vogel/dm)

80-Zeichen-Hardcopy

Für den 40-Zeichen-Modus gibt es genug Hardcopy-Programme, die man nötigenfalls auch umschreiben kann. Für den 80-Zeichen-Modus sind jedoch spezielle Hardcopy-Routinen nötig, die es aber derzeit noch nicht so häufig gibt. Mit diesem Programm »VDCHC.0C00« (Listing 7, bitte mit dem MSE im C64-Modus eingeben) erhalten Sie eine 80-Zeichen-Hardcopy-Routine mit folgenden Vorzügen:

– Die Attribute »zweiter Zeichensatz an« (Bit 7), »Revers« (Bit 6) und »Unterstrichen« (Bit 5) des Attributspeichers werden mit berücksichtigt. Sollte der Attributspeicher abgeschaltet sein (Bit 6 des VDC-Registers 25 ist Null), erkennt dies das Programm und ignoriert die Attribute.

– Veränderte Adressen für Bildschirmspeicher (Register 12), Attributspeicher (Register 20) und Zeichensatz (Register 28, Bits 5 bis 7) werden berücksichtigt. Die Hardcopy-Routine wertet die Register 13 und 21 nicht aus, da das Betriebssystem des C128 nur eine Verschiebung um ganze Speicherseiten (Page) vorsieht.

– Auch veränderte oder selbst erstellte Zeichensätze werden erkannt und ausgedruckt. Das Programm arbeitet beim Ausdruck grundsätzlich im Bitmap-Modus (640-Pixel-Grafik bei Epson-Druckern und Kompatiblen) und holt die Bitmuster-Daten nicht aus dem Zeichensatz-ROM, sondern aus dem VDC-RAM.

– Ebenfalls verarbeitet die Routine ein verändertes Bildschirmformat. Die Anzahl der dargestellten Zeilen wird aus dem Register 6 des VDC geholt, so daß alles auf dem Papier erscheint, was auf dem Bildschirm zu sehen ist. Die Anzahl der Zeichen je Zeile muß jedoch immer 80 betragen; bei anderen Formaten spielt nicht nur dieses Programm, sondern auch das Betriebssystem des C128 nicht mit, so daß diese Einschränkung nicht ins Gewicht fällt.

```
Name : vdchc.0c00             0c00 0dff
---------------------------------------
0c00 : 78 a9 2b 8d 14 03 a9 0c  e1
0c08 : 8d 15 03 58 20 7d ff 0e  f6
0c10 : d6 c4 c3 2d c8 41 52 44  47
0c18 : 43 4f 50 59 20 4d 49 54  7c
0c20 : 20 27 c1 cc d4 27 21 07  f7
0c28 : 0d 00 60 a5 d3 29 08 f0  8b
0c30 : 17 a9 65 8d 14 03 a9 fa  1d
0c38 : 8d 15 03 20 4b 0c a9 2b  27
0c40 : 8d 14 03 a9 0c 8d 15 03  55
0c48 : 4c 65 fa ad 00 ff 8d 92  16
0c50 : 0d a9 00 8d 00 ff a2 ff  6e
0c58 : b5 00 9d 00 0b ca d0 f8  b1
0c60 : a2 1c 20 da cd 29 e0 85  28
0c68 : fe a2 06 20 da cd 85 8f  8e
0c70 : a2 0c 20 da cd 85 fb a2  ba
0c78 : 14 20 da cd 85 fd a9 00  fb
0c80 : 85 fa 85 fc a2 19 20 da  ac
0c88 : cd aa 29 80 f0 03 4c 87  6c
0c90 : 0d a9 20 8d 01 0d a9 20  8b
0c98 : 8d 15 0d 8a 29 40 d0 0a  30
0ca0 : a9 2c 8d 01 0d a9 2c 8d  cd
0ca8 : 15 0d a9 7f a2 04 a0 01  6d
0cb0 : 20 ba ff a9 00 20 bd ff  5a
0cb8 : 20 c0 ff a2 7f 20 c9 ff  ad
0cc0 : 20 7d ff 1b 33 18 00 a9  49
0cc8 : 50 85 8e 20 7d ff 1b 2a  1b
0cd0 : 04 80 02 00 a2 fa 20 c4  a1
0cd8 : 0d 85 66 a2 fc 20 c4 0d  94
0ce0 : 85 67 a5 66 85 64 a9 00  71
0ce8 : 85 65 06 64 26 65 06 64  9c
0cf0 : 26 65 06 64 26 65 06 64  45
0cf8 : 26 65 a5 fe 18 65 65 85  67
0d00 : 65 20 97 0d a0 00 a2 64  5a
0d08 : 20 c4 0d 99 ef 0d c8 e6  59
0d10 : 64 c0 08 d0 f1 20 a5 0d  c1
0d18 : a2 07 a9 00 9d f7 0d ca  0c
0d20 : 10 fa a0 07 b9 ef 0d 85  11
0d28 : 68 a2 07 a5 68 3d df 0d  62
0d30 : f0 09 bd f7 0d 19 e7 0d  66
0d38 : 9d f7 0d ca 10 ed 88 10  20
0d40 : e3 a2 07 bd f7 0d 20 d2  fc
0d48 : ff ca 10 f7 a5 fa 18 69  15
0d50 : 01 85 fa a5 fb 69 00 85  9d
0d58 : fb a5 fc 18 69 01 85 fc  17
0d60 : a5 fd 69 00 85 fd a6 8e  5e
0d68 : ca 86 8e f0 03 4c d4 0c  35
0d70 : a9 0d 20 d2 ff a6 8f ca  0b
0d78 : 86 8f f0 03 4c c7 0c 20  d6
0d80 : cc ff a9 7f 20 c3 ff a2  0b
0d88 : ff bd 00 0b 95 00 ca d0  ed
0d90 : f8 a9 00 8d 00 ff 60 a5  db
0d98 : 67 29 80 f0 07 a5 65 18  35
0da0 : 69 10 85 65 60 a5 67 29  42
0da8 : 20 f0 05 a9 ff 8d f6 0d  19
0db0 : a5 67 29 40 f0 0d a2 07  6b
0db8 : bd ef 0d 49 ff 9d ef 0d  a0
0dc0 : ca 10 f5 60 8e d5 0d e8  b9
0dc8 : 8e ce 0d a2 12 a5 00 20  e4
0dd0 : cc cd a2 13 a5 00 20 cc  02
0dd8 : cd a2 1f 20 da cd 60 01  62
0de0 : 02 04 08 10 20 40 80 80  f0
0de8 : 40 20 10 08 04 02 01 00  92
0df0 : 00 00 00 00 00 00 00 00  f1
0df8 : 00 00 00 00 00 00 00 a9  4c
```

Listing 7. Hardcopy-Routine zur Darstellung der 80-Zeichen-Grafik (bitte mit dem MSE im C64-Modus eingeben)

Das Programm erkennt nicht:

– Farben auf dem Bildschirm und blinkende Darstellung. Da die Auflösung des Druckers beschränkt ist, wurde auf Grautöne zugunsten besserer Lesbarkeit und geringerem Speicherplatzverbrauch verzichtet.

– Die komplette Invertierung des Bildschirms durch Bit 6 des Registers 24, was allerdings auch eine große Belastung für das Farbband gewesen wäre.

```
80 RF=2: REM RANDFARBE
90 SCNCLR
100 PRINT "NACH BEENDIGUNG JEDER GRAPHIK
110 PRINT "WECHSELT DIE RANDFARBE.
120 PRINT "DANN TASTE DRUECKEN
130 SLEEP 5
140 GRAPHIC 1,1
150 COLOR 1,16: COLOR 0,1: COLOR 4,RF
155 X=1
160 DO UNTIL X>=44
170 CIRCLE 1,X*8,20*SIN(37*X/70)+80,30,30,,,
    ,20
180 X=X+0.7
190 LOOP
200 RF=RF+1
210 COLOR 4,RF
220 GET KEY A$
230 GRAPHIC 1,1
235 X=1
240 DO UNTIL X>=228
250 Y=Y+0.7
260 CIRCLE 1,X+30+50*SIN(X/5-0.2),Y,30,30,,,
    ,90
270 X=X+0.7
280 LOOP
290 RF=RF+1
300 COLOR 4,RF
310 GET KEY A$
320 GRAPHIC 1,1
325 X=1
330 DO UNTIL X>=230
340 CIRCLE 1,50*SIN(X/20-0.2)+X+10,X,40,20,,
    ,X,20
350 X=X+3.5
360 LOOP
370 RF=RF+1
380 COLOR 4,RF
390 GET KEY A$
400 GRAPHIC 0
```

Listing 8. Grafik-Spielereien mit dem C128

– Den Bitmap-Modus des C128. Startet man dieses Programm trotzdem, während der Bitmap-Modus aktiv ist, so erkennt dies das Programm und startet den Ausdruck gar nicht. Der Grund für das Fehlen einer Grafik-Hardcopy ist der Speicherplatz. Das Programm hätte dann in einen anderen Speicherbereich gelegt werden müssen (zum Beispiel ab $1300 oder $2000), der aber oft von anderen Programmen belegt ist.

Programm-Bedienung

Geladen und gestartet wird das Programm mit

BOOT"VDCHC.0C00"

oder

BLOAD"VDCHC.0C00":SYS 3072

Danach ist es in den Interrupt eingebunden. Eine Hardcopy erfolgt dann durch Druck auf <ALT> oder von einem Programm aus durch SYS 3147. <RUN/STOP+RESTORE> desaktiviert das Programm wieder. Ein Neustart ist mit SYS 3072 möglich.

Da die RS232-Puffer recht selten gebraucht werden, ist das Programm dort untergebracht. Es belegt den Speicherbereich von $00C00 bis $00DFE. Weiterhin wird der Kassettenpuffer als Zwischenspeicher für die Zeropage belegt, die während des Programmablaufs nach dorthin ausgelagert ist. Da der Kassettenpuffer nicht ständig benötigt wird, während dieses Programm aktiv ist, funktionieren Kassettenprogramme mit dem Hardcopy-Programm. Man muß sich aber davor hüten, während Kassettenoperationen eine Hardcopy auszulösen.

(Frank-Ch. Krügel/dm)

Grafik-Spielereien

Das kleine Programm »GRAFIKEN« (Listing 8) erstellt drei HiRes-Grafiken für den C128 mittels der Sinusfunktion und des CIRCLE-Befehls. Der Segmentwinkel 20 (bei den Figuren 1 und 3) beschleunigt das Zeichnen der Kreise erheblich, ohne dabei ihre Form zu verändern.

Durch Ändern der Parameter oder der Funktion lassen sich unzählige Variationen entwickeln.

(Stephan Hartmann/dm)

Vielseitiges RENUMBER

Die Überschrift klingt im ersten Augenblick wie ein Eigentor, da der C128 bereits mit einer RENUMBER-Routine ausgestattet ist. Diese Funktion benötigt leider selbst im FAST-Modus viel Zeit. Bei einem Programm von etwa 30 KByte Länge beispielsweise mehr als vier Minuten.

Immerhin, mit der Geschwindigkeit könnte man leben, gäbe es da nicht einen zweiten Punkt. Mit der RENUMBER-Routine des C128 läßt sich ein Programm nicht neu durchstrukturieren, da die Reihenfolge der Programmzeilen nicht verändert wird.

Die hier vorgestellte Routine hingegen ermöglicht es nicht nur, Programmabschnitte neu durchzunumerieren, sondern man kann mit ihr auch Programmteile verschieben. Zum Beispiel Unterroutinen an das Programmende. Wer viel in Basic programmiert, weiß, wieviel »Handarbeit« man sich dadurch ersparen kann.

Eingabe und Start

Bitte tippen Sie zuerst den Basic-Lader (Listing 9) ab und starten das Programm durch RUN. Jede Zeile ist mit Prüfsummen versehen. Bei fehlerhaften Zeilen wird ein »PRÜFSUMMENFEHLER IN XX« ausgegeben. Sobald der Computer alle DATA-Zeilen akzeptiert hat, speichern Sie das Programm. Starten Sie anschließend den veränderten RENUMBER-Befehl mit SYS 4888.

```
85 REM PROGRAMM-AUFRUF: SYS 4888
90 :
100 READ D$        : AD=DEC(D$)     : ZL=1001
110 FOR I=1 TO 10 : READ D$        : D=DEC(D$)
120 SU=SU+D        : POKE AD,D      : AD=AD+1
130 NEXT           : READ P$        : PR=DEC(P$)
140 D=PR-32768     : IF D=>0 THEN PR=D
150 IF PR<>SU THEN PRINT "FEHLER IN",ZL: STOP
160 ZL=ZL+1: SU=0: IF D<0 THEN 110: ELSE END
170 :
180 :
190 :
1000 DATA 1300
1001 DATA EA,EA,EA,EA,EA,EA,EA,EA,EA,EA, 0924
1002 DATA EA,EA,EA,EA,EA,EA,EA,EA,EA,EA, 0924
1003 DATA EA,EA,EA,EA,EA,A9,2D,8D,04,03, 05FC
1004 DATA A9,18,8D,05,03,A9,34,8D,08,03, 02CB
1005 DATA A9,13,8D,09,03,60,8D,03,FF,58, 039C
1006 DATA 68,AA,20,CF,18,C9,F8,F0,03,4C, 0519
1007 DATA A2,4A,8A,48,A0,07,B9,95,19,99, 0465
1008 DATA A7,19,88,10,F7,88,84,FC,84,FD, 05D8
1009 DATA A0,00,84,FA,84,FB,20,E5,18,B0, 056A
1010 DATA 27,8C,A7,19,8D,A8,19,20,E8,18, 03E1
1011 DATA B0,18,8C,A9,19,8D,AA,19,20,E8, 046E
1012 DATA 18,B0,0D,84,FA,85,FB,20,E8,18, 04F3
```

Listing 9. Ein RENUMBER-Befehl, der auch Programmbereiche verschiebt

```
1013 DATA B0,04,84,FC,85,FD,A0,FF,C6,3E, 0659
1014 DATA 20,DA,18,78,20,0F,14,8D,01,FF, 035A
1015 DATA 20,05,19,20,79,16,A5,2A,05,2B, 01EC
1016 DATA F0,6B,18,A9,04,65,2A,90,02,E6, 0427
1017 DATA 2B,0A,26,2B,0A,26,2B,85,2A,38, 01C8
1018 DATA AD,12,12,E5,2A,85,28,AD,13,12, 035F
1019 DATA E5,2B,85,29,20,6F,18,38,A5,28, 036A
1020 DATA E5,1A,85,22,A5,29,E5,1B,85,23, 041C
1021 DATA 90,4B,20,B4,17,A5,2B,06,2A,2A, 02F0
1022 DATA 06,2A,0A,AA,E8,86,26,E4,23,B0, 042F
1023 DATA 38,A5,FA,05,FB,D0,0C,20,5B,18, 0446
1024 DATA B1,20,85,FA,C8,B1,20,85,FB,20, 0589
1025 DATA 15,17,F0,0C,10,21,20,E1,16,D0, 0340
1026 DATA 1C,20,C4,15,D0,17,20,78,14,20, 02C8
1027 DATA 07,19,8D,03,FF,20,0F,14,4C,2E, 026C
1028 DATA 13,20,4F,4F,4C,82,4F,A0,80,20, 032E
1029 DATA 07,19,8D,03,FF,58,98,10,05,A2, 0356
1030 DATA 10,4C,3C,4D,B9,31,14,85,26,B9, 0347
1031 DATA 32,14,85,27,4C,84,4D,39,14,4F, 02AB
1032 DATA 14,64,14,53,54,41,52,54,20,4E, 0288
1033 DATA 55,4D,42,45,52,20,4E,4F,54,20, 02AC
1034 DATA 46,4F,55,4E,C4,4C,49,4E,45,20, 0344
1035 DATA 4E,55,4D,42,45,52,20,54,4F,4F, 02DB
1036 DATA 20,4C,41,52,47,C5,43,52,4F,53, 0342
1037 DATA 53,49,4E,47,20,4C,49,4E,45,20, 0299
1038 DATA 4E,55,4D,42,45,D2,20,6F,18,20, 0310
1039 DATA 66,18,85,18,85,1C,85,2F,18,8A, 0312
1040 DATA 85,19,65,26,85,1D,85,30,20,1A, 02BA
1041 DATA 19,20,66,18,A0,00,B1,2F,91,16, 02DE
1042 DATA C8,B1,2F,91,16,D0,07,C8,91,16, 0495
1043 DATA A8,91,16,60,C8,C8,B1,2F,88,AA, 0551
1044 DATA B1,2F,20,86,15,A0,02,91,16,C8, 03AC
1045 DATA 8A,91,16,C8,B1,2F,91,16,30,0B, 03BB
1046 DATA D0,F7,C8,20,8F,18,20,9B,18,90, 04B9
1047 DATA C7,20,14,18,D0,E9,20,70,15,90, 0401
1048 DATA E5,84,33,A2,00,95,18,E8,C8,B1, 054C
1049 DATA 2F,20,79,15,B0,F5,84,34,CA,B5, 04B9
1050 DATA 18,85,24,A0,00,84,25,F0,1A,B5, 03C9
1051 DATA 18,85,27,C8,10,0F,18,A5,24,79, 0305
1052 DATA 9D,19,85,24,A5,25,79,A2,19,85, 03E2
1053 DATA 25,C6,27,10,ED,CA,10,E3,20,8A, 0476
1054 DATA 15,B0,56,F0,54,85,24,86,25,A2, 0455
1055 DATA 04,A9,30,85,27,38,B0,06,84,24, 031F
1056 DATA 85,25,E6,27,A5,24,FD,9D,19,A8, 04DB
1057 DATA A5,25,FD,A2,19,B0,ED,A5,27,95, 0580
1058 DATA 18,CA,D0,DF,18,A5,24,69,30,85, 0490
1059 DATA 18,AC,34,00,20,8F,18,A4,33,20, 02B6
1060 DATA 9B,18,A2,04,B5,18,C9,30,D0,03, 03F2
1061 DATA CA,D0,F7,A0,00,B5,18,91,16,C8, 056D
1062 DATA CA,10,F8,20,9B,18,A0,00,2C,A4, 0415
1063 DATA 33,4C,BC,14,C8,B1,2F,91,16,C9, 0467
1064 DATA 20,F0,F7,38,E9,30,90,07,C9,0A, 04C2
1065 DATA B0,02,38,60,18,60,85,24,86,25, 0316
1066 DATA 84,27,20,5B,18,A0,00,F0,05,C8, 039B
1067 DATA D0,02,E6,21,B1,20,AA,C8,B1,20, 04ED
1068 DATA C9,FA,B0,1A,C8,C8,E4,24,D0,EB, 06E0
1069 DATA C5,25,D0,E7,B1,20,AA,88,B1,20, 0575
1070 DATA A4,27,C5,24,D0,02,E4,25,18,60, 0407
1071 DATA A5,24,A6,25,A4,27,38,60,A5,FA, 0496
1072 DATA 85,24,A5,FB,85,25,20,B5,16,85, 0463
1073 DATA 2F,86,30,A6,FC,A4,FD,E8,D0,01, 05E1
1074 DATA C8,86,24,84,25,20,B5,16,85,31, 03BC
1075 DATA 86,32,38,E5,2F,85,35,8A,E5,30, 045D
1076 DATA 85,36,38,A5,22,E5,35,A5,23,E5, 0481
1077 DATA 36,B0,03,A0,04,60,AD,A7,19,85, 03DF
1078 DATA 24,AD,A8,19,85,25,20,B5,16,85, 03AC
1079 DATA 33,86,34,85,18,86,19,18,65,35, 02DB
1080 DATA 85,1C,8A,65,36,85,1D,20,6F,18, 030F
1081 DATA 18,65,35,85,37,8A,65,36,85,38, 0350
1082 DATA 20,1A,19,38,A5,2F,E5,33,A5,30, 034C
1083 DATA E5,34,90,1A,18,A5,2F,65,35,85, 03CE
1084 DATA 2F,A5,30,65,36,85,30,18,A5,31, 0342
1085 DATA 65,35,85,31,A5,32,65,36,85,32, 0379
1086 DATA A2,05,B5,2F,95,18,CA,10,F9,20, 042B
1087 DATA 1A,19,A5,31,85,18,A5,32,85,19, 031B
1088 DATA A5,37,85,1A,A5,38,85,1B,A5,2F, 03CC
1089 DATA 85,1C,A5,30,85,1D,20,1A,19,20, 028B
1090 DATA 66,18,A9,00,85,2A,85,2B,A0,01, 0327
1091 DATA B1,16,F0,28,E6,2A,D0,02,E6,2B, 04D2
1092 DATA A0,03,C8,B1,16,D0,FB,C8,18,98, 0575
1093 DATA 65,16,A6,17,90,01,E8,A0,00,91, 03E2
1094 DATA 16,48,C8,8A,91,16,68,85,16,86, 03E0
1095 DATA 17,4C,82,16,A0,00,91,16,60,20, 02C2
1096 DATA 66,18,D0,05,20,D2,16,F0,0D,A0, 03F8
1097 DATA 02,B1,16,AA,C8,B1,16,20,B3,18, 03ED
1098 DATA 90,EE,A5,16,A6,17,18,60,A0,00, 040E
1099 DATA B1,16,AA,C8,B1,16,F0,04,86,16, 0490
1100 DATA 85,17,60,20,5B,18,AD,A7,19,85, 0381
1101 DATA 24,AD,A8,19,85,25,20,A7,18,70, 038B
1102 DATA 1C,F0,1D,B0,05,20,80,18,90,F2, 0418
1103 DATA AD,AB,19,85,24,AD,AC,19,85,25, 0436
1104 DATA 20,A7,18,70,04,90,05,F0,03,A0, 037B
1105 DATA 00,60,A0,06,60,20,5B,18,A5,FA, 0398
1106 DATA 85,24,A5,FB,85,25,A9,00,AA,8E, 04D4
1107 DATA AD,19,8D,AE,19,20,A7,18,70,4D, 03B6
1108 DATA F0,07,B0,49,20,80,18,90,EC,A5, 04C9
1109 DATA FC,85,24,A5,FD,85,25,AD,A7,19, 055E
1110 DATA 85,16,AD,A8,19,85,17,A0,03,A5, 03ED
1111 DATA 17,8D,AC,19,91,20,88,A5,16,8D, 03EA
1112 DATA AB,19,91,20,18,6D,A9,19,85,16, 0357
1113 DATA AD,AA,19,65,17,85,17,B0,15,C9, 0416
1114 DATA FA,B0,11,20,80,18,20,A7,18,70, 03C2
1115 DATA 1F,90,D2,F0,D0,B0,06,A0,02,60, 04F9
1116 DATA A0,04,60,AD,AB,19,85,24,AD,AC, 0477
1117 DATA 19,85,25,20,A7,18,70,04,90,1C, 02C2
1118 DATA F0,1A,AD,AE,19,AE,AD,19,D0,03, 04C5
1119 DATA A8,F0,12,CD,A8,19,90,0D,D0,05, 04AA
1120 DATA EC,A7,19,90,06,A0,CF,60,A0,D5, 0586
1121 DATA 60,A0,00,60,20,66,18,20,5B,18, 0291
1122 DATA 84,2A,84,2B,A0,01,B1,16,D0,0A, 039F
1123 DATA A9,FF,A0,04,91,20,88,10,FB,60, 04F0
1124 DATA A0,03,B1,16,AA,88,B1,16,A0,00, 0403
1125 DATA 91,20,C8,48,8A,91,20,C8,68,91, 04BD
1126 DATA 20,C8,8A,91,20,20,80,18,A0,03, 037E
1127 DATA C8,B1,16,F0,1D,10,F9,20,14,18, 03F1
1128 DATA D0,F4,C8,B1,16,F0,11,C9,20,F0, 062D
1129 DATA F7,20,79,15,90,E7,E6,2A,D0,E2, 05DE
1130 DATA E6,2B,D0,DE,20,D2,16,4C,BE,17, 04E8
1131 DATA C9,89,90,14,F0,12,C9,D6,B0,0E, 0555
1132 DATA C9,D5,B0,0A,C9,A7,B0,06,C9,8D, 05D4
1133 DATA B0,02,C9,8A,60,20,0D,43,A2,FF, 0476
1134 DATA E8,BD,00,02,F0,22,C9,20,F0,F6, 0588
1135 DATA C9,30,90,04,C9,3A,90,16,A2,FF, 04D7
1136 DATA E8,BD,00,02,D0,FA,BD,00,02,9D, 04CD
1137 DATA 01,02,CA,10,F7,A9,3A,8D,00,02, 0346
1138 DATA 60,A0,00,A5,28,85,20,A5,29,85, 03C5
1139 DATA 21,60,A5,2D,A6,2E,85,16,86,17, 035F
1140 DATA 60,18,AD,10,12,AE,11,12,69,02, 0283
1141 DATA 90,01,E8,85,1A,86,1B,60,18,48, 0379
1142 DATA A9,04,65,20,85,20,90,02,E6,21, 0370
1143 DATA 68,18,60,18,98,65,2F,85,2F,90, 0368
1144 DATA 03,E6,30,18,60,18,98,65,16,85, 0341
1145 DATA 16,90,03,E6,17,18,60,A0,00,B1, 036F
1146 DATA 20,AA,C8,B1,20,C9,FB,B0,0C,C5, 05A8
1147 DATA 25,90,06,F0,02,B0,02,E4,24,B8, 041F
1148 DATA 60,2C,94,19,60,C8,20,D1,18,C9, 0433
1149 DATA 20,F0,F8,60,A0,00,2C,A0,01,8D, 0462
1150 DATA 01,FF,B1,3D,8D,03,FF,60,18,98, 048D
1151 DATA 65,3D,85,3D,90,02,E6,3E,60,A0, 041A
1152 DATA 01,2C,A0,FF,20,C3,18,C9,00,F0, 0480
1153 DATA 13,C9,3A,F0,0F,C9,2C,D0,01,C8, 04A3
1154 DATA 20,DA,18,20,D7,77,20,15,88,18, 0355
1155 DATA 60,18,24,38,A2,20,B5,18,90,05, 02F8
1156 DATA BD,AF,19,95,18,9D,AF,19,CA,10, 0471
1157 DATA F1,60,38,A5,1A,E5,18,A8,A5,1B, 04AD
1158 DATA E5,19,48,38,A5,18,E5,1C,AA,A5, 048B
1159 DATA 19,E5,1D,D0,05,8A,D0,02,68,60, 0414
1160 DATA 68,90,31,AA,E8,84,1A,38,A9,00, 043A
1161 DATA E5,1A,A8,84,1A,38,A5,18,E5,1A, 0439
1162 DATA 85,18,B0,02,C6,19,38,A5,1C,E5, 040C
1163 DATA 1A,85,1C,B0,02,C6,1D,B1,18,91, 03AA
1164 DATA 1C,C8,D0,F9,E6,19,E6,1D,CA,D0, 0649
1165 DATA F2,60,84,18,AA,E8,18,65,1D,85, 049F
1166 DATA 1D,38,A5,1A,E5,18,85,1A,B0,02, 0362
1167 DATA C6,1B,98,F0,07,B1,1A,91,1C,88, 0470
1168 DATA D0,F9,B1,1A,91,1C,C6,1B,C6,1D, 0505
1169 DATA CA,D0,F2,60,40,64,00,0A,00,00, 039A
1170 DATA 00,00,00,01,0A,64,E8,10,00,00, 0167
1171 DATA 00,03,27,00,00,00,00,00,00,00, 802A
```

Listing 9. Ein RENUMBER-Befehl, der auch Programmbereiche verschiebt (Schluß)

Bedienungshinweise

Wir gehen ab jetzt immer vom folgenden Fall aus: Sie haben ein Basic-Programm im Speicher stehen, das zehn Zeilen umfaßt, das in Hunderterschritten, beginnend bei 50, numeriert ist. Geben Sie nun einfach RENUMBER ein. Das Programm wurde in Zehnerschritten, beginnend bei Zeile 100, numeriert. Wenn auf den RENUMBER-Befehl keine weiteren Angaben folgen, wählt das Programm die voreingestellten Werte »100,10« (Start-Zeilennummer, Schrittweite).

Haben Sie das gedachte Programm zuvor in Zehnerschritten bearbeitet, geben Sie nun einmal RENUMBER 500,1,150 ein. Hier werden die Zeilennummern ab 150 neu in Einerschritten numeriert. Die ehemaligen Programmzeilen, die ab Zeile 150 in Zehnerschritten numeriert bestanden, beginnen nun bei Zeile 500 in Einerabständen.

Mit dem folgendem Befehl RENUMBER 1,1,160,190 passiert folgendes: Da die Zeilen 160 bis 190 am Programmende sehr deplaziert dastehen würden, findet man sie in Einerschritten numeriert am Programmanfang wieder. Somit lassen sich also beliebige Programmblöcke verschieben. Die komplette Syntax ist also RENUMBER <Startadr. neu>, <Schrittweite>, <(Startadr.alt>, <Endadresse alt>.

(Heino Velder/dm)

Register- und Speicherbereich-Retter

Einige werden bestimmt schon gelegentlich versucht haben, an den Speicherinhalt von Programmen zu kommen, um zum Beispiel eine Hardcopy eines Grafikbildschirms zu machen.

Das Programm (Listing 10) tippen Sie bitte ab und speichern es. Nach dem Start mit RUN wird ein ablauffähiges Maschinenprogramm auf Diskette geschrieben. Dieses Machinenprogramm können Sie nun immer mit

```
BLOAD"MEMORY-RETTER"
```

laden und durch SYS 4864 starten. Der C128 meldet sich nun im C64-Modus, der einen ganz normalen Eindruck macht. Es scheint aber nur so, denn der C64-Modus wurde in RAM-BANK 1 (Variablen-BANK) gestartet. Außerdem wurden noch einige Kopier-Routinen installiert. Nun kann man das Programm, das man analysieren möchte, laden und starten. Sobald man nun den Reset-Knopf drückt, treten die Kopier-Routinen in der anderen RAM-BANK (BANK 0) in Aktion. Der komplette Ein-/Ausgabebereich wird nach $D000 bis $DFFF in BANK 0 kopiert. Außerdem überträgt das Programm Page 2 und 3 ($200 bis $3FF) nach $2200 bis $23FF in BANK 0. Der restliche Speicher des C64-Modus bleibt bis auf die obersten 256 Byte unverändert in BANK 1.

Nun kann man in aller Ruhe die VIC-Register untersuchen und feststellen, in welchem Bereich sich der Bildschirm oder die Sprites befanden.

Funktionsweise

In BANK 1 gibt es bei $FFF5 3 Byte, die den Code »CBM« enthalten. Bei einem Reset prüft das Betriebssystem diese Bytes. Stimmt der Code, so wird der Inhalt der Bytes $FFF8 bis $FFF9 als Sprungadresse in BANK 0 genommen. Das Startprogramm verbiegt nun diesen Vektor auf die Kopier-Routine. Danach schaltet es RAM-BANK 1 und den C64-Modus ein. Außerdem wird auch der VIC auf diese BANK gelegt, da sonst der Bildschirm dunkel bliebe.

Da sich der C64-Modus nun in BANK 1 befindet, könnte ein Programm den »CBM«-Code bei $FFF5 zerstören. Um dies zu verhindern, wird der gemeinsame Bereich (Common Aera) der BANKs zum Speicherende nach $FC00 bis $FFFF geschoben. Greift ein C64-Programm auf diesen Bereich zu, so wird immer BANK 0 adressiert, der »CBM«-Code ist sicher. Allerdings greift der VIC weiterhin auf BANK 1 zu, so daß es bei Programmen, die zum Beispiel den Grafikbildschirm nach $E000 bis $FFFF legen, zu Wirrwar im unteren Bildschirmbereich kommt. Dies ist jedoch harmlos, da die Grafik trotzdem intern korrekt aufgebaut wird. Nach einem Reset sitzt alles wieder an der richtigen Stelle.

(Ulrich Textoris/dm)

Zeichensatz kopieren

Das Programm »Schnellkopierer« (Listing 11) kopiert den gesammten Zeichensatz ($D000 bis $E000) in einen angegebenen Bereich. Im Byte 53272 geben die Bits 1 bis 3 den Platz des Zeichensatzes an. Daraus ergeben sich acht mögliche Adressen, ab denen der Zeichensatz im Speicher liegen kann:

0, 2048, 4096, 6144, 8192, 10240, 12288, 14336

Dabei bildet der erste Fall eine Ausnahme, da sich dort die Zeropage befindet. Im Einschaltzustand liegt der Zeichensatz ab 4096 ($1000).

Das eigentlich interessante am Programm ist, daß das Maschinenprogramm (ohne SYS-Aufruf) den Zeichensatz in 0,07 Sekunden an die angegebene Stelle kopiert. Ein Basic-Programm würde 1000mal mehr Zeit benötigen (70 Sekunden). Abschließend muß der Basic-Bereich noch vor dem Überschreiben geschützt werden. (Thorsten Schüngel/dm)

```
10 FOR A=1 TO 384: READ B: C=C+B: NEXT
20 IF ABS(C-46629)>1E-4 THEN PRINT "PRUEFSUM
   MENFEHLER": STOP
30 OPEN 2,8,1,"0:MEMORY-RETTER"
40 PRINT#2,CHR$(0);CHR$(19);: REM STARTADRES
   SE
50 RESTORE
60 FOR A=1 TO 384: READ B: PRINT#2,CHR$(B);:
    NEXT
70 CLOSE 2: PRINT "LADER IST FERTIG"
80 END
32000 DATA 120,169,0,141,0,255,169,0,133,252
      ,169,253,133,253,169,252,141,185,2
32001 DATA 160,0,162,1,185,176,19,32,119,255
      ,32,119,255,200,208,242,169,255,133
32002 DATA 253,162,1,160,248,169,0,32,119,25
      5,162,1,200,169,20,32,119,255,234
32003 DATA 234,234,169,1,141,33,208,169,96,1
      33,252,169,19,133,253,169,252,141
32004 DATA 185,2,160,0,162,1,185,96,19,32,11
      9,255,200,208,245,234,234,234,234
32005 DATA 234,234,120,169,227,133,1,169,47,
      133,0,169,64,141,0,255,169,72,141
32006 DATA 6,213,169,240,141,8,213,169,0,141
      ,7,213,169,247,141,5,213,234,234,234
32007 DATA 108,252,255,0,255,162,0,160,0,140
      ,6,213,181,0,72,189,0,1,72,189,0,2
32008 DATA 72,189,0,3,160,7,140,6,213,234,15
      7,0,35,104,157,0,34,104,157,0,33,169
32009 DATA 0,141,245,255,169,126,141,0,255,1
      62,0,160,0,140,6,213,181,0,72,189
32010 DATA 0,1,72,189,0,2,72,189,0,3,160,7,1
      40,6,213,234,157,0,35,104,157,0,34
32011 DATA 104,157,0,33,104,157,0,32,232,208
      ,213,160,4,140,6,213,96,0,0,0,0,0
32012 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,169,0
      ,141,1,213,169,63,141,2,213,162,16
32013 DATA 169,0,133,252,169,208,133,253,160
      ,0,141,1,255,177,252,141,2,255,145
32014 DATA 252,200,208,243,230,253,202,208,2
      38,141,1,255,234,169,0,133,4,169,253
32015 DATA 133,3,8,104,133,5,169,1,133,2,32,
      110,255,169,7,141,6,213,169,127,141
32016 DATA 3,213,162,0,141,2,255,189,0,252,1
      41,3,255,157,0,252,141,2,255,189,0
32017 DATA 253,141,3,255,157,0,253,141,2,255
      ,189,0,254,141,3,255,157,0,254,141
32018 DATA 2,255,232,208,214,169,4,141,5,213
      ,108,252,255,0,0,0
```

Listing 10. »Memory-Retter«

```
10 DATA 165,147,74,74,133,148,169,0,133,146,
   173,14,220,41,254,141,14,220
20 DATA 165,1,41,251,133,1,169,0,133,20,169,
   208,133,21,160,0,177,20,145,146
30 DATA 200,208,249,230,21,230,147,165,21,20
   1,224,208,239,165,1,9,4,133,1
40 DATA 173,14,220,9,1,141,14,220,173,24,208
   ,41,240,5,148,141,24,208,96
50 FOR T=49152 TO 49227: READ A: POKE T,A: N
   EXT
60 INPUT "ADRESSE :";B: B=INT(B/256)
70 POKE 147,B
80 SYS 49152
90 PRINT : PRINT "BASIC-BEREICH MUSS NOCH":
   PRINT "GESCHUETZT WERDEN !!!": END
100 REM ******************************
110 REM **** MOEGLICHER BEREICH: ****
120 REM ****                     ****
130 REM **** (0),2048,4096,6144, ****
140 REM ****   8192,10240,       ****
150 REM ****         12288,14336  ****
160 REM ******************************
```

Listing 11. Kopiert den Zeichensatz in nur 0,07 Sekunden an eine andere Stelle

```
10 GRAPHIC 1,1: GRAPHIC 0
20 FOR A=64128 TO 64216: BANK 15: B=PEEK(A):
    BANK 0: POKE 16295+A-64128,B: NEXT
30 POKE 832,16295-INT(16295/256)*256
40 POKE 833,INT(16295/256)
50 POKE 16294+74,69: POKE 16294+75,70
60 KEY 1,"A": KEY 3,"B": KEY 5,"C": KEY 7,"D
   "
```

Listing 12. Umfunktionierung des Zehnerblocks zu einer Hex-Tastatur

Zehnerblock als Hex-Tastatur

Mit diesem Programm (Listing 12) läßt sich eine Hex-Tastatur auf dem Zehnerblock simulieren. Dies ist vor allem bei der Eingabe von Hex-Dumps von Vorteil.

Nach dem Starten mit RUN kopiert das Programm die Tastenbelegung der Normaltastatur in den Bereich, der normalerweise von der Grafik benutzt wird. Der Schutz vor Überschreiben durch ein Basic-Programm erfolgt durch GRAPHIC 1,1 (Reservieren des Grafikspeichers). Dann wird in den Zeilen 30 und 40 der Zeiger, der auf die Tastaturtabelle der geSHIFTeten Tastatur deutet, auf die entsprechenden Werte der neuen Tabelle gelenkt. Zeile 50 weist den Tasten 74 und 75 (+ und -) einen neuen ASCII-Code zu. Abschließend erfolgt die Neudefinition der Funktionstasten.

A	B	C	D
7	8	9	E
4	5	6	F
1	2	3	E N
0		.	T

Bild 1. Die geSHIFTete Hex-Belegung des Zehnerblocks

Um mit der neuen Tastaturbelegung arbeiten zu können, muß <SHIFT> oder <SHIFT/LOCK> gedrückt beziehungsweise eingerastet sein.

Benötigen Sie den Grafikbildschirm, dann können Sie die Zahl 16259 (Anfangsadresse der Tabelle) im ganzen Programm durch die Adresse ersetzen, die Ihnen am geeignetsten erscheint. Bild 1 zeigt Ihnen die neue Belegung des geSHIFTeten Zehnerblocks. (Christian Maul/dm)

Sichere Input-Routine

Die herkömmliche INPUT-Routine des C 128 steht in keinem Verhältnis zu seinen anderen Fähigkeiten. Diese Routine (Listing 13) behebt diese Mängel. Nun kann nicht mehr eine Eingabemaske zerstört werden, indem man mit dem Cursor aus der Zeile fährt.

Die Eingabe-Routine wird durch

```
PA$="S1Z1S2Z2MINSC":GOSUB 200
```

aufgerufen. Dabei besitzen die zu übergebenden Parameter folgende Bedeutung:

```
S1 (2 Zeichen) - Spaltennummer der linken oberen
Ecke des Eingabefensters
Z1 (2 Zeichen) - Zeilennummer der linken oberen
Ecke des Eingabefensters
S2 (2 Zeichen) - Spaltennummer der rechten unteren
Ecke des Eingabefensters
Z2 (2 Zeichen) - Zeilennummer der rechten unteren
Ecke des Eingabefensters
MIN (3 Zeichen) - minimale Eingabelänge
S (1 Zeichen) - Wert = 1: Leerzeichen nicht erlaubt
                Wert = 0: Leerzeichen ist erlaubt
C (1 Zeichen) - Codenummer für zulässige Zeichen
                Wert = 1: nur Groß-/Kleinbuchstaben
                und Leerzeichen
                Wert = 2: Groß-/Kleinbuchstaben,
                Zahlen, Leerzeichen, Bindestrich
                und Punkt
                Wert = 3: Groß-/Kleinbuchstaben,
                Zahlen und SPACE
                Wert = 4: Zahlen, Bindestrich und
                Schrägstrich
                Wert = 5: Zahlen, Punkt und
                Apostroph
```

Zum Anfügen eines weiteren Codes ist folgendes zu tun: In Zeile 320 hinter dem Befehl »ON C GOSUB 350,360,370,380,390« ist »,400« anzufügen. Danach ist in Zeile 400 eine Bedingung ähnlich der in den Zeilen 350 bis 390 einzugeben.

Selbstverständlich können auch die bereits vorhandenen Codes verändert werden.

Nach dem Aufruf steht die Eingabe dann in der Variable »IN$«, wobei Leerzeichen am Ende der Eingabe nicht abgeschnitten werden. (Alexander Niepel/dm)

Komfortabler DATAwandler

Der sonst recht ordentliche Monitor des C 128 läßt einen Befehl zum Umwandeln von Speicherbereichen in Basic-DATA-Zeilen vermissen. Die folgende Routine (Listing 14) behebt dieses Manko.

Das Programm muß unbedingt vor dem ersten Start gespeichert werden, da es sich anschließend selbst zerstört.

Das Programm ist durch

```
RUN"DATAWANDLER"
```

zu laden und zu starten. Nach kurzer Wartezeit erscheint das Hauptmenü, in dem folgende Angaben erwartet werden:

– Adressen hexadezimal oder dezimal angeben ?

```
100 FAST : PRINT CHR$(14) CHR$(147) CHR$(5);
110 PRINT "NAME{10SPACE}(1-30 ZEICHEN, A-Z){9SPACE
    }:" :
    PA$="440073000101" : GOSUB 200 : NA$=IN$
120 PRINT "STRASSE{7SPACE}(1-30 ZEICHEN, A-Z 0-9 .
     -) :" :
    PA$="440173010O102" : GOSUB 200 : ST$=IN$
130 CHAR ,0,2,"WOHNORT{7SPACE}(1-30 ZEICHEN, A-Z 0
    -9){5SPACE}:" :
    PA$="440273020O103" : GOSUB 200 : WO$=IN$
140 CHAR ,0,3,"TELEFONNUMMER (1-20 ZEICHEN, 0-9 -
    /){5SPACE}:" :
    PA$="440363030O114" : GOSUB 200 : TE$=IN$
150 CHAR ,0,4,"GEBURTSDATUM{2SPACE}(6-10 ZEICHEN,
    0-9 . '){5SPACE}:" :
    PA$="440453040O615" : GOSUB 200 : GE$=IN$
160 CHAR ,0,5,"GEBURTSORT{4SPACE}(1-20 ZEICHEN, A-
    Z){9SPACE}:" :
    PA$="440563050O101" : GOSUB 200 : GT$=IN$
170 CHAR ,0,8,"IHRE DATEN:{2SPACE}"+NA$ : CHAR ,13
    ,9,ST$ : CHAR ,13,10,WO$ :
    CHAR ,13,11,"TEL.:"+TE$ : CHAR ,13,12,"GEB.: A
    M "+GE$ : CHAR ,19,13,
    "IN "+GT$+AN$
180 END
190 REM ***************************** "INPUT-ROUTIN
    E" ***************************
200 FAST : E$=CHR$(27): S1=VAL(LEFT$(PA$,2)): Z1=V
    AL(MID$(PA$,3,2)):
    S2=VAL(MID$(PA$,5,2)): Z2=VAL(MID$(PA$,7,2)):
    MI=VAL(MID$(PA$,9,3)):
    CO=VAL(RIGHT$(PA$,1)): AU=32
210 SP=VAL(MID$(PA$,12,1)): WINDOW S1,Z1,S2,Z2: PR
    INT CHR$(28)E$"M"UE$ CHR$(19);:
    MA=(S2-S1+1)*(Z2-Z1+1): IN$=UE$: PO=1: UE$="":
     AU$=E$+"U": AN$=E$+"F"+E$+"U":
    CU=2603
220 PRINT AN$;: GET KEY N$: N=ASC(N$): POKE CU,AU:
     L=LEN(IN$)
230 IF N=13 THEN BEGIN : IF EI=0 AND L>=MI THEN PR
    INT E$E$E$"L" CHR$(5) CHR$(19) CHR$(19)
    AU$: POKE CU,AU: RETURN : ELSE 220: BEND
240 IF N=19 THEN BEGIN : IF PO<>1 AND EI=0 THEN
    PRINT E$E$N$;: PO=1: GOTO 220: ELSE 220: BEND
250 IF N=147 THEN EI=0: PO=1: IN$="": PRINT E$E$N$
    ;: GOTO 220
260 IF N=29 THEN BEGIN : IF EI=0 AND PO<=L AND PO<
    MA THEN PO=PO+1: PRINT E$E$N$;: GOTO 220:
    ELSE 220: BEND
270 IF N=157 THEN BEGIN : IF EI=0 AND PO>1 THEN PO
    =PO-1: PRINT E$E$N$;: GOTO 220:
    ELSE 220: BEND
280 IF N=17 THEN BEGIN : IF EI=0 AND PO+S2-S1+1<=M
    A AND PO+S2-S1+1<L+2 THEN PO=PO
    +S2-S1+1: PRINT E$E$N$;: GOTO 220: ELSE 220: B
    END
290 IF N=20 THEN BEGIN : IF PO>1 THEN IN$=LEFT$(IN
    $,PO-2)+RIGHT$(IN$,L-PO+1): L=L-1:
    PO=PO-1: PRINT E$E$N$;: GOTO 220: ELSE 220: BE
    ND
300 IF N=145 THEN BEGIN : IF EI=0 AND PO-S2+S1-1>0
     THEN PO=PO-S2+S1-1: PRINT E$E$N$;:
    GOTO 220: ELSE 220: BEND
310 IF N=148 THEN BEGIN : IF L<MA AND PO<=L THEN L
    =L+1: IN$=LEFT$(IN$,PO-1)+" "+RIGHT$
    (IN$,L-PO): EI=EI+SP: PRINT E$E$N$;: GOTO 220:
     ELSE 220: BEND
320 ON CO GOSUB 350,360,370,380,390: IF FL=1 THEN
    FL=0: GOTO 220
330 PRINT E$E$N$;: IF PO=L+1 THEN IN$=IN$+N$: L=L+
    1: ELSE MID$(IN$,PO,1)=N$
340 PO=PO+1+(PO=MA): EI=EI-1-(EI=0): GOTO 220
350 FL=1: IF((N>64 AND N<91) OR(N>192 AND N<219) O
    R N=32) AND PO<=MA THEN FL=0:
    RETURN : ELSE RETURN
360 FL=1: IF((N>47 AND N<58) OR(N>64 AND N<91) OR(
    N>192 AND N<219) OR N=45 OR N=32 OR N=46)
    AND PO<=MA THEN FL=0: RETURN : ELSE RETURN
370 FL=1: IF((N>47 AND N<58) OR(N>64 AND N<91) OR(
    N>192 AND N<219) OR N=32) AND PO<=MA THEN
    FL=0: RETURN : ELSE RETURN
380 FL=1: IF((N>46 AND N<58) OR N=45) AND PO<=MA T
    HEN FL=0: RETURN : ELSE RETURN
390 FL=1: IF((N>47 AND N<58) OR N=46 OR N=39) AND
    PO<=MA THEN FL=0: RETURN : ELSE RETURN
```

Listing 13. Verbesserung des INPUT-Befehls

Als Antwort drücken Sie entweder die Taste <H> für hexadezimal oder <D> für dezimal.

– Start- und Endadresse:

Hier wird die tatsächliche Start- und Endadresse eingegeben. Das Programm berücksichtigt, ob die Eingaben hexadezimal (vier mögliche Zeichen) oder dezimal (fünf mögliche Zeichen) erfolgen sollen.

– Erste Zeilennummer:

Es muß die allererste Zeilennummer des zu erzeugenden Programms angegeben werden. Sie muß größer oder gleich 300 sein, da der Teil des DATA-Wandlers, der sich während der Umwandlung noch im Speicher befindet, alle Zeilennummern bis 290 belegt.

Die Eingabe kann mit <RETURN> abgebrochen werden. Die Eingabe erfolgt auf jeden Fall dezimal.

– Schrittweite:

Die Schrittweite bestimmt den Abstand zwischen den einzelnen Zeilennummern. Sie muß größer Null und kleiner 99 sein. Wird eine Null eingegeben oder nur <RETURN> gedrückt, nimmt das Programm als Schrittweite den Wert 1 an. Auch hier erfolgt die Eingabe auf jeden Fall dezimal.

– DATAs hexadezimal oder dezimal ausgeben?

Hier genügt als Antwort <H> oder <D>. Es ist jedoch zu bedenken, daß hexadezimale Zahlen weniger Speicherplatz benötigen (pro auszulesende Speicherstelle ein Byte weniger). Das ist durch die Schreibweise bedingt. Bei der dezimalen Ausgabe werden alle Werte der besseren Übersicht wegen als dreistellige Zahlen ausgegeben. Durch Löschen der Befehlskette

T$=RIGHT$("000"+T$,3)

```
10 SCNCLR : PRINT "{2HOME}": WINDOW 0,0,39,2
   4.1
20 GOSUB 300
30 REM " *****{2SPACE}EINLESEN UND SCHREIBEN
    DER DATAS{2SPACE}*****
40 FAST : IF SW=0 THEN SW=1: BANK SB
50 IF AF$="H" THEN RE$="R$": HE$="DEC(R$)":
   SY$="DEC("+CHR$(34)+HEX$(B)+CHR$
   (34)+")": ELSE RE$="R": HE$="R": SY$=STR$
   (B)
60 PRINT "{CLR} DELETE 290-": PRINT : PRINT
   :  PRINT "GOTO 70": GOTO 220
70 Z1$=STR$(Z)+" FAST:BANK"+STR$(SB): PRINT
   "{CLR}"Z1$
80 Z=Z+S: Z2$=STR$(Z)+" FORA="+STR$(B)+"TO"+
   STR$(E)+":READ"+RE$+":
   POKE A,"+HE$+": NEXT ":PRINTZ2$
90 Z=Z+S: Z3$=STR$(Z)+" SLOW: SYS "+SY$: PRI
   NT Z3$: Z=Z+S: PRINT "GOTO 110"
100 GOTO 210
110 BZ=0
120 SCNCLR :
130 PRINT Z;"DATA";
140 IF AF$="D" THEN 150: ELSE 160
150 FOR D=B TO B+7: T=PEEK(D): T$=STR$(T): T
    $=RIGHT$(T$,LEN(T$)-1): T$=RIGHT$
    ("000"+T$,3): PRINT T$;",";: NEXT : GOTO
     170
160 FOR J=B TO B+7: T$=RIGHT$(HEX$(PEEK(J)),
    2): PRINT T$;",";: NEXT
170 PRINT CHR$(20)
180 Z=Z+S: B=B+8: BZ=BZ+1
190 IF BZ<7 AND B< E+1 THEN GOTO 130
200 IF B < E+1 THEN PRINT "GOTO 110": ELSE S
    LOW : PRINT "DELETE -280"
210 PRINT : PRINT
220 POKE 842,19
230 FOR TP=843 TO 843+9
240 POKE TP,13
250 NEXT
260 POKE 208,10
270 END
280 SLOW
```

Listing 14. Ein komfortabler DATA-Wandler

```
290 REM "{2SPACE}*****{3SPACE}AUFBAU DES MEN
    UE'S{2SPACE}*****
300 FAST : SCNCLR : COLOR 0,1 : COLOR 4,1:
310 PRINT "{GREY1,RVSON}M{38SPACE}N {GREY3,R
    VOFF}";
315 PRINT "UDDDDDDDDDDDDDDDDDDDDDDDDDDDD
    DDDDI{RVSON,GREY1} "
320 FOR L=1 TO 20
330 PRINT "{GREY1,RVSON} {RVOFF,GREY3}G{36SP
    ACE}H{GREY1,RVSON} "
340 NEXT :
350 PRINT "{GREY1,RVSON} {RVOFF,GREY3}JFFFFF
    FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFK{RVSON,G
    REY1} ";
355 PRINT "N{38SPACE}M"
360 PRINT "{HOME,WHITE,3DOWN,3RIGHT}1) ADRES
    SEN HEXADEZIMAL ODER{15RIGHT}
    DEZIMAL EINGEBEN (H/D)
370 PRINT "{DOWN,3RIGHT}2) STARTADRESSE{7SPA
    CE}:"
380 PRINT "{DOWN,3RIGHT}3) ENDADRESSE{9SPACE
    }:"
390 PRINT "{DOWN,3RIGHT}4) ERSTE ZEILENNUMME
    R:"
400 PRINT "{DOWN,3RIGHT}5) SCHRITTWEITE{7SPA
    CE}:"
410 PRINT "{DOWN,3RIGHT}6) DATAS HEXADEZIMAL
     ODER{18RIGHT}
    DEZIMAL AUSGEBEN (H/D)."
420 PRINT "{DOWN,3RIGHT}7) SPEICHERBANK (0-1
    5):"
430 SLOW : PRINT : PRINT : PRINT
440 REM "{2SPACE}*****{3SPACE}MENUE ABFRAGE{
    2SPACE}*****
450 TRAP 1130: FOR C=1 TO 7: PRINT "{WHITE,U
    P,3RIGHT}MEINE ANTWORT AUF{8SPACE,8LEFT}
    ";C;"{RED}"
460 ON C GOSUB 510,600,700,800,900,1000,1100
470 NEXT
480 PRINT "{WHITE,UP,3RIGHT}EINGABEN OKAY ?{
    2SPACE}(J/N)": GET KEY E$: IF E$<>"J" AN
    D E$<>"N"
    THEN 480: ELSE IF E$="J" THEN GOTO 490:
    ELSE 300
490 PRINT "{HOME,21DOWN}": FOR A=1 TO 19:
    PRINT "{2UP,3RIGHT,34SPACE,3RIGHT}";: NE
    XT :
    WINDOW 2,2,37,21: GOTO 40
500 REM "{2SPACE}*****{2SPACE}ADRESSEN HEX.
    ODER DEZ. EINGEBEN ?{2SPACE}*****
510 POKE 55485,2: GET KEY A1$: IF A1$<>"H" A
    ND A1$<>"D" THEN 510: ELSE HD$=A1$:
    IF HD$="H" THEN POKE 1213,8: RETURN : EL
    SE POKE 1213,4: RETURN
520 :
530 :
540 :
550 :
560 :
570 :
580 :
590 REM "{2SPACE}*****{3SPACE}STARTADRESSE{3
    SPACE}*****
600 ST$="": IF HD$="H" THEN G=4: ELSE G=5
610 PRINT "{UP}" TAB(23): FOR K=1 TO G: GET
    KEY M$(K): ST$=ST$+M$(K): PRINT ;
    M$(K);: NEXT : IF HD$="H" THEN 620: ELSE
     630
620 B=DEC(ST$): PRINT "{HOME,6DOWN}";TAB(26)
    ;"$" HEX$(B);B:
    FOR V=1 TO 14: PRINT : NEXT : RETURN
630 B=VAL(ST$): PRINT "{HOME,6DOWN}";TAB(26)
    ;"$" HEX$(B);B:
    FOR V=1 TO 14: PRINT : NEXT : RETURN
640 :
650 :
660 :
670 :
680 :
690 REM "{2SPACE}*****{3SPACE}ENDADRESSE{3SP
    ACE}*****
700 ST$="": IF HD$="H" THEN G=4: ELSE G=5
710 PRINT "{UP}" TAB(23): FOR K=1 TO G: GET
    KEY M$(K): ST$=ST$+M$(K): PRINT ;
    M$(K);: NEXT : IF HD$="H" THEN 720: ELSE
     730
720 E=DEC(ST$): PRINT "{HOME,8DOWN}";TAB(26)
    ;"$" HEX$(E);E: FOR V=1 TO 12:
    PRINT : NEXT : RETURN
730 E=VAL(ST$): PRINT "{HOME,8DOWN}";TAB(26)
    ;"$" HEX$(E);E: FOR V=1 TO 12:
    PRINT : NEXT : RETURN
740 :
750 :
760 :
770 :
780 :
790 REM "{2SPACE}*****{4SPACE}ERSTE ZEILENNU
    MMER{4SPACE}*****
800 ZE$="": PRINT "{UP}" TAB(23)"{5SPACE,5LE
    FT}";: FOR K=1 TO 5: GET KEY Z$: IF Z$=C
    HR$(13)
    THEN 810: ELSE ZE$=ZE$+Z$: PRINT ;Z$;: N
    EXT
810 Z=VAL(ZE$): IF Z<30 THEN PRINT : GOTO 80
    0: ELSE PRINT "{HOME,10DOWN}" TAB(26)Z:
    FOR V=1 TO 10: PRINT : NEXT : RETURN
820 :
830 :
840 :
850 :
860 :
870 :
880 :
890 REM *****  SCHRITTWEITE  *****
900 SW$="": PRINT "{UP}" TAB(23)"{5SPACE,5LE
    FT}";: FOR K=1 TO 2: GET KEY S$: IF S$=C
    HR$(13)
    THEN 910: ELSE SW$=SW$+S$: PRINT S$;: NE
    XT
910 S=VAL(SW$): PRINT "{HOME,12DOWN}" TAB(26
    )S: FOR V=1 TO 8: PRINT : NEXT : RETURN
920 :
930 :
940 :
950 :
960 :
970 :
980 :
990 REM "{2SPACE}*****{2SPACE}DATAS HEX. ODE
    R DEZ. AUSGEBEN{2SPACE}*****
1000 POKE 55925,2: GET KEY A1$: IF A1$<>"H"
     AND A1$<>"D" THEN 1000: ELSE AF$=A1$:
     IF AF$="H" THEN POKE 1653,8: RETURN : E
     LSE POKE 1653,4: RETURN
1010 :
1020 :
1030 :
1040 :
1050 :
1060 :
1070 :
1080 :
1090 REM "{2SPACE}*****{3SPACE}SPEICHERBANK{
     3SPACE}*****
1100 SB$="": PRINT "{UP}" TAB(23)"{5SPACE,5L
     EFT}";: FOR K=1 TO 2: GET KEY B$: IF B$
     =CHR$(13)
     THEN 1110: ELSE SB$=SB$+B$: PRINT B$;:
     NEXT
1110 SB=VAL(SB$): IF SB>15 THEN PRINT : GOTO
      1100: ELSE PRINT "{HOME,17DOWN}"TAB
1115 (26)SB: FOR V=1 TO 3: PRINT : NEXT : RE
     TURN
1120 REM "{2SPACE}*****{4SPACE}FEHLERBEHANDL
     UNG{4SPACE}*****
1130 TRAP 1130: FZ=EL
1140 FZ=INT(FZ/10): PRINT "{5LEFT,5SPACE,5LE
     FT}"
1150 ON FZ-5 GOTO 600,700,800,900,1000,1100
```

Listing 14. Ein komfortabler DATA-Wandler (Schluß)

in Zeile 150 ist dieses jedoch zu umgehen.

– Speicherbank:

Die Speicherbank, aus der gelesen wird, kann alle Werte zwischen 0 und 15 annehmen. Die Eingabe erfolgt auf jeden Fall dezimal.

Während der Menüabfrage wird bei einem Eingabefehler sofort in eine Fehlerbehandlungsroutine gesprungen, die aber nur solche Fehler abfängt, welche vom Betriebssystem erkannt werden (SYNTAX ERROR oder OVERFLOW ERROR). Damit die Fehlerbehandlung funktioniert, müssen auch die Zeilen mit den Doppelpunkten abgetippt werden. Dies ist folgendermaßen zu begründen: In der Variable EL steht die Zeilennummer, in der der Fehler aufgetreten ist. Der Vorkommateil des Zehntels von EL wird mit zehn multipliziert. Man erhält so die erste Zeilennummer des Unterprogrammes, in dem der Fehler aufgetreten ist, wohin dann verzweigt wird. REM-Zeilen werden durch Doppelpunkte ersetzt.

Sind die Eingaben beendet und bestätigt, werden die Zeilen 290 bis 1150 gelöscht, damit das zu erstellende Programm bei Zeilennummer 300 beginnen kann.

Wenn Sie schon auf dem C 64 mit DATA-Wandlern gearbeitet haben, werden Sie sich sicher darüber wundern, daß bei diesem Programm nicht unter jedem Zeilenblock Zeilennummer, Schrittweite, momentane Adresse und Endadresse stehen. Der Grund dafür ist in der verschiedenen Speicherform für Variable zu suchen. Der C 64 speichert die Variablen direkt hinter dem Basic-Programm. Die Folge ist, daß sie von jeder neuen Zeile überschrieben werden. Der C 128 legt die Variablen deshalb in einer separaten Speicherbank ab. Sie werden deshalb nicht gelöscht.

(Christian Maul/dm)

```
10 REM UNDEFINIEREN DER F-TASTEN 1-10
20 REM  F 9 =  SHIFT+RUN/STOP
30 REM  F10 =  HELP-TASTE
40 REM  VON FRANK PROBST IM MAI '86
50 :
60 A$="GO64"+CHR$(13)
70 F=10                   : REM F-TASTENNUMMER
80 :
90 BANK 1                 : REM VARIABLENBANK
100 SD=POINTER(A$)        : REM ADRESSE DES STR
    INGDISCRIPTORS
110 L=PEEK(SD)            : REM = L=LEN(A$)
120 POKE 252,PEEK(SD+1):  REM LO-BYTE DES STR
    INGZEIGERS
130 POKE 253,PEEK(SD+2):  REM HI-BYTE DES STR
    INGZEIGERS
140 POKE 254,1            : REM BANKNUMMER
150 :
160 BANK 15
170 SYS 65381,252,F,L     : REM AUFRUF VON PFKE
    Y ($FF65)
```

Listing 15. Neue Funktion für <SHIFT+RUN/STOP>

Zehn Funktionstasten belegen

Wer hat beim Eintippen eines Programms nicht schon mal <SHIFT+RUN/STOP> gleichzeitig gedrückt. Liegt zu allem Unglück noch eine Diskette im eingeschalteten Diskettenlaufwerk, erlebt man einige schreckliche Sekunden. Das erste Programm wird mit DLOAD"*" geladen und überschreibt ein im Speicher befindliches. Kein noch so guter OLD-Befehl mindert das Drama.

```
10 REM ****************************
20 REM **          U H R         **
30 REM **       H. TETENS        **
40 REM **(C)*************31.12.85*
50 :
60 GRAPHIC 0,1: COLOR 0,6: COLOR 1,2
70 INPUT "BITTE UHRZEIT (STUNDE.MINUTE)";T$
80 INPUT "UND NOCH DAS HEUTIGE DATUM{2RIGHT}
   31.12.1985{12LEFT}";D$
90 IF LEN(D$)<>10 THEN 80: ELSE GOSUB 440
100 IF MID$(T$,2,1)="." THEN T$="0"+T$
110 TI$=LEFT$(T$,2)+MID$(T$,4,2)+"00"
120 PRINT CHR$(27)"X"
130 GRAPHIC 3,1: COLOR 0,10: COLOR 1,1: COLO
    R 2,3: COLOR 3,6: COLOR 4,1: COLOR 5,2
140 MX=93: MY=69
150 BOX 1,2,2,158,198: BOX 1,4,4,156,196:
              REM AUSSENRAND
160 BOX 1,12,120,53,142,,1: BOX 1,12,158,112
    ,183,,1:   REM AUSMAHL
170 BOX 1,10,118,55,144:   BOX 1,10,156,114,
    185:        REM AUSMAHLRAND 1
180 BOX 1,8,116,57,146:    BOX 1,8,154,116,1
    87:         REM AUSMAHLRAND 2
190 CHAR 1,4,16,"00:00:00",1
200 CHAR 1,4,21,D$,1
210 CIRCLE 1,MX,MY,56,56: CIRCLE 1,MX,MY,49,
    49: PAINT 1,MX-51,MY,1
220 CIRCLE 1,MX,MY,46,46: CIRCLE 1,MX,MY,6,6
    : PAINT 1,MX,MY-10,1
230 : DIM X(60): DIM Y(60)
240 : FOR I=0 TO 59: J=6*I*π/180: X(I)=SIN(J
    ): Y(I)=COS(J)
250 :  DRAW 2,MX+Y(I)*44,MY+X(I)*44 TO MX+Y(
    I)*50,MY+X(I)*50
260 :  IF I/5=INT(I/5) THEN DRAW 3,MX+Y(I)*4
    4,MY+X(I)*44 TO MX+Y(I)*54,MY+X(I)*54
270 : NEXT
280 :
290 H1=MX: H2=MX: H3=MX: K1=MY: K2=MY: K3=MY
300 T2=VAL(MID$(TI$,3,2)): T=VAL(LEFT$(TI$,2
    )): T1=(T-INT(T/12)*12)*5+INT(T2/15)
310 TT$=LEFT$(TI$,2)+":"+MID$(TI$,3,2)+":"+R
    IGHT$(TI$,2): CHAR 1,4,16,TT$,1
320 IF TI$="000000" THEN GOSUB 450: CHAR 1,4
    ,21,D$,1
330 IF T1<>U1 THEN  BOX 1,H1-1,K1-18,H1+1,K1
    +18,L1,1: H1=MX+X(T1)*10: K1=MY-Y(T1)*10
    : L1=T1*6
340 IF T2<>U2 THEN  BOX 1,H2-1,K2-25,H2+1,K2
    +25,L2,1: H2=MX+X(T2)*17: K2=MY-Y(T2)*17
    : L2=T2*6
350 DRAW 1,MX+X(U3)*(-8),MY-Y(U3)*(-8) TO MX
    +X(U3)*43,MY-Y(U3)*43
360 BOX 3,H1-1,K1-18,H1+1,K1+18,L1,1
370 BOX 3,H2-1,K2-25,H2+1,K2+25,L2,1
380 DRAW 2,MX+X(T3)*(-8),MY-Y(T3)*(-8) TO MX
    +X(T3)*43,MY-Y(T3)*43
390 IF T1<>U1 THEN PLAY "T9 O4 QAFG HC QCGA
    WF": IF T1/5=INT(T1/5) THEN
    FOR I=1 TO T1/5: PLAY "T9 O4 QGE WC": NE
    XT
400 U1=T1: U2=T2: U3=T3
410 T3=VAL(RIGHT$(TI$,2)): IF T3=U3 THEN 410
    : ELSE 300
420 :
430 REM        * * * WOCHENTAGBERECHNUNG UND D
    ATUMSFORMATIERUNG * * *
440 DT=VAL(LEFT$(D$,2)): DM=VAL(MID$(D$,4,2)
    ): DJ=VAL(RIGHT$(D$,4)): GOTO 460
450 DT=DT+1: IF(DT=29 AND DM=2) OR(DT=31 AND
    (DM=4 OR DM=6 OR DM=9 OR DM=11)) OR
    DT=32 THEN DT=1: DM=DM+1: IF DM=13 THEN
    DM=1: DJ=DJ+1
460 WT=INT(365.25*(DJ+(DM<3)))+INT(30.6*(DM+
    1-12*(DM<3)))+DT-621049
470 WU=INT((WT/7-INT(WT/7))*7+0.5): RESTORE
480 DATA SONNTAG,MONTAG,DIENSTAG,MITTWOCH,DO
    NNERSTAG,FREITAG,SAMSTAG
490 FOR I=0 TO WU: READ DT$: NEXT
500 D$=LEFT$(DT$+"{8SPACE}",13)+RIGHT$(STR$(
    DT),2)+"."+RIGHT$(STR$
    (DM),2)+"."+RIGHT$(STR$(DJ),4): RETURN
```

Listing 16. Darstellung einer Analog/Digital-Uhr auf dem Bildschirm

```
10 REM *********************************
20 REM **    3*DIRECTORY-PRINT         **
30 REM **     H. T E T E N S           **
40 REM **(C)****************31.5.1986*
50 :
60 PRINT "{CLR,DOWN} {RVSON,4SPACE}* * *{2SP
   ACE}H.{2SPACE}T E T E N S{3SPACE}* * * (C
   ){RVOFF}"
70 PRINT "{2DOWN} DIESES PROGRAMM DRUCKT DAS
   DIRECTORY"
80 PRINT "{3SPACE}DES C-64 ODER C-128 SLOW+F
   AST (!)"
90 PRINT "{DOWN} 3-/5-SPALTIG IN INDEX+SCHMA
   LSCHRIFT"
100 G=RWINDOW(2): IF G=80 THEN 160
110 PRINT "{DOWN} (1) 40-ZEICHEN-MODUS"
120 PRINT " (2) 80-ZEICHEN-MODUS: DOPPELTE G
    ESCHW.": POKE 198,0
130 : GET KEY G$: IF G$<"1" OR G$>"2" THEN 1
    30
140 : IF G$="1" THEN 500: ELSE PRINT "{DOWN,
    RVSON}80-ZEICHEN-MODUS EIN{RVOFF}, DANN
    RETURN"
150 : INPUT G$: IF G$="" THEN 100
160 : FAST
500 DIM F$(200): GOSUB 3000: REM ---DRUCKER-
    AUSWAHL, SCHRIFT-AUSWAHL
510 PRINT "{3DOWN,RVSON}MENU:{RVOFF}"
520 PRINT " (0) GELESENE DISKETTE NOCHMAL DR
    UCKEN"
530 PRINT " (1) NAECHSTE DISK LESEN, 1 MAL D
    RUCKEN"
540 PRINT " (2) NAECHSTE DISK LESEN, 2 MAL D
    RUCKEN"
550 PRINT " (3) NAECHSTE DISK LESEN, 3 MAL D
    RUCKEN"
560 PRINT " (4) NAECHSTE DISK LESEN, 4 MAL D
    RUCKEN"
570 PRINT " 8D) DRUCKER, SCHRIFT AENDERN"
580 PRINT " (E) ENDE": G$="": POKE 198,0
590 GET G$: IF G$="E" THEN SLOW : END
600 : IF G$="D" THEN GOSUB 3000: GOTO 510
610 : IF G$="" OR G$<"0" OR G$>"9" THEN 590
620 : IF G$="0" THEN F9=1: GOTO 2020
630 F9=VAL(G$)
640 :
1000 : OPEN 1,8,0,"$": REM * * * * * * * LES
     EROUTINE * * * * * * *
1010 :  PRINT "{DOWN,RVSON}";
1020 :  GET #1,A$,A$,A$,A$,A$,A$,A$
1030 :  IF ST<>0 THEN PRINT "{RVSON}>>>KEIN
     DIRECTORY VORHANDEN<<<{RVOFF}": CLOSE 1
     : GOTO 510
1040 :  A=0: FOR I=0 TO 6: F$(I)="{2SPACE}":
      NEXT
1050 :     FOR I=LEN(F$(A)) TO 25
1060 :      GET #1,A$: IF A$=CHR$(34) THEN 1
     080
1070 :      F$(A)=F$(A)+A$
1080 :     NEXT : PRINT F$(A): A=A+1
1090 :  GET #1,A$: IF A$<>"" THEN 1090
1100 :  GET #1,A$,B$: IF A$="" AND B$="" THE
    N 1140
1110 :  GET #1,A$,B$: C$=STR$(ASC(A$)+ASC(B$
     )*256): F$(A)=RIGHT$("{3SPACE}"+C$+" ",
     5)
1120 :  FOR I=0 TO 5-LEN(C$): GET #1,A$: NEX
    T I
1130 :  GOTO 1050
1140 : CLOSE 1
1150 :
2000 FZ$=C$+" BLOCK FREI.": F$(A-1)=".": B=A
    -1
2010 IF B/S<>INT(B/S) THEN B=B+1: GOTO 2010
2020 : OPEN 1,4,(F0-1): REM     * * * * * DR
    UCKERROUTINE * * * * *
2030 :  IF F1=1 THEN PRINT#1,CHR$(27) CHR$(1
    5);
2040 :  IF F2=1 THEN PRINT#1,CHR$(27) CHR$(8
    3) CHR$(1);CHR$(27) CHR$(65) CHR$(5);
2050 :    FOR I=1 TO F9
2060 :     PRINT#1,"** H.TETENS **";CHR$(14)
    ;F$(0);CHR$(20);FZ$
2070 :      FOR J=1 TO 79: PRINT#1,"=";: NEX
    T J: PRINT#1
2080 :        FOR K=1 TO B/S: N$=""
2090 :         FOR L=0 TO S: PRINT#1,N$;F$(
    K+L*B/S);: N$=" ": NEXT L: PRINT#1
2100 :        NEXT K: IF F9>1 THEN PRINT#1:
     PRINT#1: PRINT#1
2110 :    NEXT I
2120 :  PRINT#1,CHR$(27) CHR$(64)
2130 : CLOSE 1
2140 GOTO 510
2150 :
3000 PRINT "{3DOWN}BITTE " CHR$(18)"DRUCKER"
     CHR$(146)" EINSCHALTEN,{2SPACE}WAEHLEN
    :"
3010 PRINT " (1) FX-80 EPSON": PRINT " (2) F
    + RITEMAN": POKE 198,0
3020 : GET KEY G$: IF G$>="1" AND G$<="2" TH
    EN F0=VAL(G$): ELSE GOTO 3020
3030 : IF F0=1 THEN PRINT "{DOWN,RVSON}FX-80
     -SCHRIFT:{RVOFF}": ELSE PRINT "{DOWN,R
    VSON}F+ -SCHRIFT:{RVOFF}"
3040 PRINT " (1) NORMAL-SCHRIFT, DIN A-4, 3-
    SPALTIG"
3050 PRINT " (2) INDEX-SCHRIFT,{2SPACE}DIN A
    -4, 3-SPALTIG"
3060 PRINT " (3) SCHMAL-SCHRIFT, DISK,{4SPAC
    E}3-SPALTIG"
3070 PRINT " (4) INDEX + SCHMAL, DISK,{4SPAC
    E}3-SPALTIG"
3080 PRINT " (5) SCHMAL-SCHRIFT, DIN A-4, 5-
    SPALTIG"
3090 PRINT " (6) INDEX + SCHMAL, DIN A-4, 5-
    SPALTIG": POKE 198,0
3100 : GET KEY G$: IF G$<"1" OR G$>"6" THEN
    3100
3110 : G=VAL(G$): S=3: IF G>=5 THEN S=5: G=G
    -2      : REM 3/5-SPALTIG
3120 :         F1=0: IF G>=3 THEN F1=1: G=G
    -2    : REM SCHMAL
3130 :         F2=0: IF G>=2 THEN F2=1
       : REM SCHMAL
3140 RETURN
```

Listing 17. Drucken Sie sich Ihre Disketten-Directories, die Sie auf die Diskettenhüllen kleben können

Der C 128 hat belegbare Funktionstasten. Jedoch sind es derer nicht acht, sondern zehn. Die Kombination <SHIFT+RUN/STOP> stellt gewissermaßen <F9> und die Taste <HELP> die zehnte Funktionstaste dar. Mit dem KEY-Befehl des Basic 7.0 lassen sich aber nur <F1> bis <F8> belegen, so daß man <F9> und <F10> nicht verändern kann.

Doch existiert eine Betriebssystem-Routine mit dem Namen »PFKEY«. Diese wird vom Basic-Interpreter zur Funktionstastenänderung benutzt. PFKEY unterstützt aber alle zehn Funktionstasten. So geht man nun mit PFKEY um:

Die Zeichenkette, die den Funktionstastentext enthält, irgendwo im Speicher ablegen. Das LOW-Byte der Startadresse des Strings muß in eine Speicherzelle der Zeropage geschrieben werden. Ebenso müssen das High-Byte und die BANK-Nummer in den nachfolgenden Speicherzellen stehen. Ein Beispiel:

```
$13000  47 4F 36 34 0D 00 00 00  GO64m
```




```
** H.TETENS **     VERSUCHE PX                               PX  2A 684 BLOCK FREI.
 4 PYSTUMPF           PRG   18 26NUT WIDERSTAND PRG    3 45 RECHTECK ZEI   PRG
 2 WUERFEL NEIPP      PRG    7  27 QUADR.FKT.H  PRG    3 45A RECHTECK ZEI  PRG
 2 1 QUADER SCHNEI    PRG    8 27 RECHT.D.ECK   PRG    5 49 F+SCHRIFTEN    PRG
 6 2 QUADER M.KOMM.   PRG    1 28 FIBONACCI     PRG    6 49B SEQ.FILE W/R  PRG
 1 3 FUNKT.PLOT TE    PRG    5 29 SORTIEREN     PRG    2 50 KOPFRECHNEN    PRG
 5 4 QUAD.+LIN.GL.N   PRG    6 30 SORTIEREN TE  PRG   39 EHMANN FAST KOMP  PRG
 3 5 KESTUMPF N       PRG    3 31 ZAHLENRATEN   PRG    7 49C SEQ.FILE W/R  PRG
 2 ZEITKONST.KOND     PRG   15 32 SPEISENPARODI PRG    2 51 PRIMZAHLEN     PRG
 2 7ZEITKONST..KOND   PRG    7 33 TRANSISTOREN  PRG   35 54 HANG           PRG
 2 8TRAFO-MESSEN      PRG    8 33.1 VARIATION   PRG    6 53 VERSUCH HU     PRG
 3 10 C-ZEITKONST.    PRG    2 34 LOTTO         PRG    2 5                PRG
 4 9 KONDENS.AUFB.    PRG    4 35A SORT 1       PRG    7 35E SORT EHMANN   PRG
 2 11 PUNKTE-NOTEN    PRG    4 35B SORT 2       PRG   11 56 AUFG.10/4 MO   PRG
 2 13 BENZINPREIS     PRG    4 35C SORT 3       PRG    1 55 AUFG.10/1 MO   PRG
25 12 KSTU HU         PRG    1 35D SORT ZENSUR  PRG    8 54B KOPFRECHNEN   PRG
 2 14 VERKUPFERUNG    PRG   10 KLASS.LIST HO    PRG    8 54B KOPFRECH TE   PRG
 1 15 HERON TE        PRG    4 37MO-TIERE-SORT  PRG    3 57 INPUT 128      PRG
 3 16 BEGIN BEND      PRG   10 36KLASS.LIST HO  PRG   47 KIPPSCHALTUNGEN   PRG
 9 17 C-ZEITKONST.    PRG   14 NOTEN HU         PRG    1 58 CRSR MALT      PRG
 2 18 TANNENBAUM TE   PRG   28 36T KLASSENLIST  PRG   15 61 RATENSPAREN    PRG
 2 19 DIVISIONSL.     PRG    2 38 GRAPHIC VERS. PRG    2 60 ORTELT GB      PRG
12 17 MIT FORTO       PRG   15 7A SPEISP.       PRG   11 63 DIRECTORY-PRI  PRG
14 17.1 MIT FORTO     PRG    3 11A PUNKT-NOT/SC PRG    7 59 SQUASH         PRG
 1 22 FUNKT.PLOT RC   PRG    4 49 SEQ.FILE W/R  PRG   12 64 DIRECTORY-PRI  PRG
 1 21 FUNKT.PLOT TE   PRG    2 39 RIESENBUCHST  PRG    9 48C1 SORT AG      PRG
 5 24 ZINSESZINS TE   PRG    6 40 READ DATA UEB PRG   13 48F SORT-COM-AG   PRG
 5 23 ZINSESZINS TE   PRG   13 41 TRANSISTOR HO PRG   12 3A DIRECTORY-PRI  PRG
 8 20 LEHRER 7        PRG    4 42 SEQ.FILE TEST PRG    5 70 NIMM           PRG
12 COM.19 SYNTAX 10   PRG   10 43 MASTERMIND    PRG
 5 25 <KO-TANNE HO    PRG    1 44 F+ DRUCK-TEST PRG
```

Bild 2. Ausdruck eines Disketten-Directories auf einem Epson-Drucker oder Kompatiblen

```
10 GRAPHIC 1,1
20 DO UNTIL E=180
30 E=E+4
40 CIRCLE 1,159,99,E,,,,E,72
50 LOOP
100 OPEN 1,4
110 FOR T=312 TO 0 STEP -8
120 DRAW 1,T,0 TO T,199
130 SSHAPE A$,T,0,T+7,199
140 A$=LEFT$(A$,200)
150 PRINT#1,CHR$(8)+A$
160 NEXT
170 PRINT#1,CHR$(15)
180 CLOSE 1
```

Listing 18. Hardcopy-Routine für den C128

Jetzt muß die Adresse in die Zeropage geschrieben werden:

`$000FB  00 30 01 ...`

Der nächste Schritt ist das Laden des Akkus mit der Zeropage-Adresse. Für das Beispiel wäre das:

`LDA #$FB`

Nun noch die gewählte Funktionstastennummer in das X-Register:

`LDX #$0A (für die HELP-Taste)`

und die Länge des Textes in das Y-Register:

`LDY #$05 (G064 + <RETURN> = 5 Zeichen)`

Abschließend in die Routine PFKEY springen:

`JMP $FF65 beziehungsweise JSR $FF65`

Listing 15 erledigt das alles für Sie in Basic. Die Variable F enthält die Funktionstastennummer, A$ den Tastentext.

(Frank Probst/dm)

Digital/Analog-Uhr

Will man mal die Zeit wissen, muß auf die Uhr gesehen werden. Diese Software-Uhr (Listing 16) zeigt einem eine HiRes-Uhr schön groß, farbig und sowohl analog mit Stunden-, Minuten- und Sekundenzeiger als auch digital mit Stunden, Minuten und Sekunden. Zusätzlich »schlägt« die Computer-Uhr alle Viertelstunde und zu jeder vollen Stunde die Anzahl der Stunden.

Nach dem Start durch RUN muß erst die aktuelle Zeit eingegeben werden (Eingabe in Stunden und Minuten). Danach möchte das Programm das Datum des jeweiligen Tages wissen. Den Wochentag berechnet sich das Programm selbst (Zeilen 430 bis 500). Ebenfalls berücksichtigt wird, ob der Monat 28, 30 oder 31 Tage hat. Für die Wochentagsberechnung werden auch die Schaltjahre berücksichtigt.

(Hauke Tetens/dm)

Directory-Ausdruck

Ordnung ist das halbe Leben. Aber wie soll man auf welchen Disketten welche Dateien finden, wenn es deren zu viele gibt? Natürlich, man kann sich ja das Directory ausdrucken lassen und dann auf die Diskettenhülle kleben, eventuell die Rückseite der Diskette noch daneben.

Doch Epson-Drucker (und Kompatible) können auch klein und schmal drucken, nur geht das beim Directory leider nicht so ohne weiteres und hätte auch nicht das richtige Format.

Dafür benötigt man also nun ein Programm, das diese Probleme löst. Mit Listing 17 können Sie selbst, sofern Sie im Besitz eines Epson-Druckers oder Kompatiblen sind, so schöne Disketten-Directories wie aus Bild 2 ersichtlich, ausdrucken.

Einmal mit RUN gestartet, erklärt sich dieses Programm selbst. Andere Druckertypen können leicht an das Programm angepaßt werden.

(Hauke Tetens/dm)

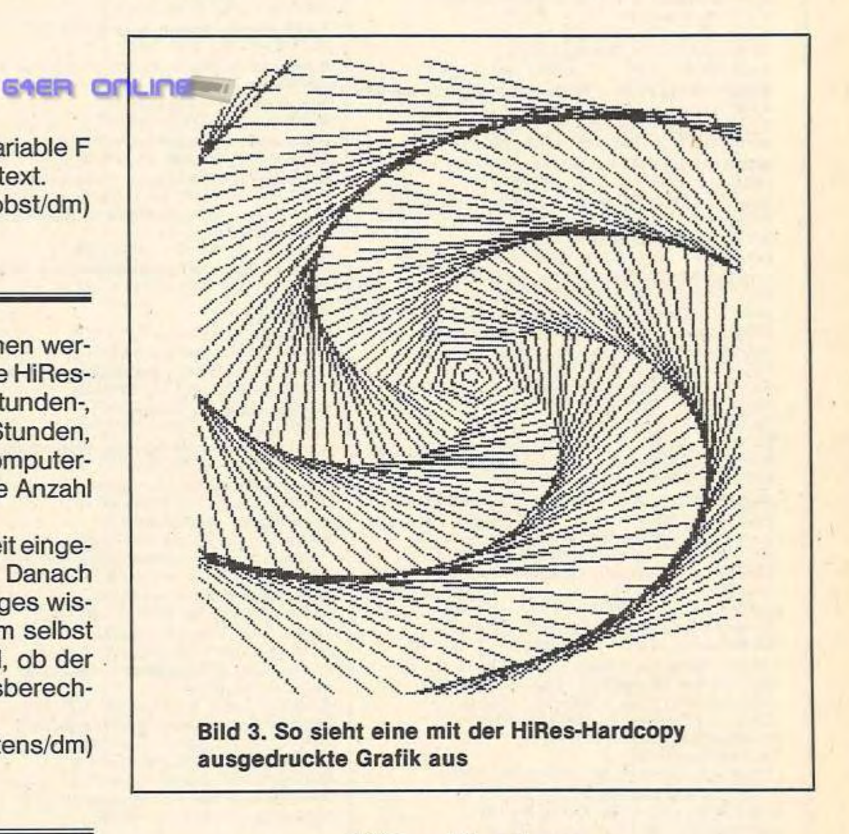


Bild 3. So sieht eine mit der HiRes-Hardcopy ausgedruckte Grafik aus

HiRes-Hardcopy

Das kleine Programm (Listing 18) gibt auf jedem herkömmlichen, grafikfähigen Drucker eine Hardcopy des ersten Grafikbildschirms aus. Die Zeilen 10 bis 60 gehören nicht dazu, sie dienen nur dazu, eine Grafik als Demo zu zeichnen (Bild 3). Die eigentliche Routine beginnt erst ab Zeile 100.

(Helmut Mestrovic/dm)

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Michael Scharfenberger

Stellv. Chefredakteur: Albert Absmeier

Leitender Redakteur: Georg Klinge

Redaktion: Herbert Buckel (bj), Roland Fieger (rf), Achim Hübner (ah), Norbert Jungmann (nj), Gottfried Knechtel (kn), Dieter Mayer (dm), Harald Meyer (hm), Markus Ohnesorg (og), Karsten Schramm (ks)

Titelfoto: Jens Jancke

Titelgestaltung: Heinz Rauner Grafik-Design

Layout:
Leo Eder (Ltg.), Sigrid Kowalewski (Cheflayouterin), Rolf Raß, Katja Milles

Produktionsleiter: Klaus Buck

Anzeigenverkaufsleitung: Ralph-Peter Rauchfuss

Anzeigenverkauf: Helmut Distl (398)

Auslandsrepräsentation:

Schweiz: Markt&Technik Vertriebs AG,
Kollerstr. 3, CH-6300 Zug,
Tel. 042-41 56 56, Telex: 862329

USA: M&T Publishing Inc.; 501 Galveston Drive
Redwood City, CA 94063
Telefon: (415) 366-3600

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt & Technik Verlag AG verlegten Publikationen und dazu, daß Markt & Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Marketingleiter: Hans Hörl (114)

Vertriebsleiter: Helmut Grünfeldt (189)

Anzeigenverwaltung und Disposition: Michaela Hörl

Verlagsleiter M&T-Buchverlag: Günther Frank (212)

Druck: SOV St. Otto-Verlag GmbH,
Laubanger 23, 8600 Bamberg

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 4613-249. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 14,–

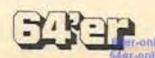
Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1, Telefon (0711) 6483-0

Verantwortlich:
Für redaktionellen Teil: Michael Scharfenberger
Für Anzeigen: Britta Fiebig

Redaktionsdirektor: Michael M. Pauly

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt & Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, 8013 Haar bei München,
Telefon (089) 4613-0, Telex 5-22052






THAI BOXING